# **СОДЕРЖАНИЕ**

[Введение 7](#_Toc38123146)

[1 Аналитический обзор существующих методов и средств по выбору   
лекарственных препаратов 8](#_Toc38123147)

[1.1 Анализ предметной области 8](#_Toc38123148)

[1.2 Обзор аналогов 9](#_Toc38123149)

[1.3 Требования к проектируемому программному обеспечению 14](#_Toc38123151)

[2 Архитектура программного обеспечения 15](#_Toc38123152)

[2.1 Описание построения веб приложения 15](#_Toc38123153)

[2.2 Описание подхода построения SPA 17](#_Toc38123153)

[2.3 Описание паттерна promise 18](#_Toc38123157)

[2**.**4 Node.js платформа для выполнения javascript на серверной стороне 19](#_Toc38123157)

[2**.**5 Выбор средств разработки 19](#_Toc38123157)

[2**.**6 Экспертные системы 21](#_Toc38123157)

[2**.**7 Функциональная модель программного комплекса 23](#_Toc38123157)

[2**.**8 Архитектура системы 24](#_Toc38123157)

[2**.**9 Информационная модель 25](#_Toc38123157)

[3 Программная реализация системы 32](#_Toc38123152)

[3**.**1 Настройка рабочего окружения и создание проекта 32](#_Toc38123157)

[3**.**2 Структура и способ построения приложения 33](#_Toc38123157)

[3**.**3 Алгоритмическое обеспечение 37](#_Toc38123157)

[3**.**4 Интерфейс приложения 38](#_Toc38123157)

[4 Верификация и опытная эксплуатация программного копмлекса 47](#_Toc38123152)

[5 Экономическое обоснование дипломной работы 54](#_Toc38123152)

[5**.**1 Расчет общей трудоемкости разработки программного обеспечения 54](#_Toc38123157)

[5**.**2 Расчет затрат на разработку программного продукта 60](#_Toc38123157)

[5**.**3 Формирование цены при создании программного обеспечения 69](#_Toc38123157)

[5**.**4 Расчет эффекта от внедрения программного обеспечения 70](#_Toc38123157)

[6 Охрана труда и техника безопасности. Защита от электромагнитных излучений 74](#_Toc38123152)

[7 Ресурсо- и энергосбережение 77](#_Toc38123158)

[Заключение 80](#_Toc38123158)

[Список использованных источников 81](#_Toc38123159)

[Приложение А Исходный код программы 82](#_Toc38123159)

[Приложение Б Руководство программиста 145](#_Toc38123159)

[Приложение В Руководство системного администратора 146](#_Toc38123159)

Перечень условных обозначений и сокращений

В настоящей пояснительной записке применяются следующие термины, обозначения и сокращения.

Сервер – программный компонент вычислительной системы, выполняющий сервисные функции по запросу клиента, предоставляя ему доступ к определенным ресурсам.

Клиент – это аппаратный или программный компонент вычислительной системы, посылающий запросы серверу.

Аутентификация – процедура проверки подлинности.

Авторизация – предоставление определенному лицу или группе лиц прав на выполнение определенных действий; а также процесс проверки (подтверждения) данных прав при попытке выполнения этих действий.

БД – база данных.

ЭВМ – электронно-вычислительная машина.

ПО – программное обеспечение.

UML – унифицированный язык моделирования (Unified Modeling Language).

SPA – одностраничное приложение (Single Page Application).

REST – передача состояния представления (Representational State Transfer).

API – программный интерфейс приложения (Application Programming Interface).

Введение

Здоровье – это самое важное составляющее, каждого человека. Сегодня, когда у человека на счету каждая минута, часто люди не обращают внимание на состояние своего здоровья, что со временем может привести к непоправимым последствиям. Также многие люди, при возникновении каких-либо проблем со здоровьем, не знают что предпринять и вынуждены обратиться к специалисту, даже если это элементарная простуда.

Данное приложение предоставляет экспертную помощь в лечении различных заболеваний. Приложение помогает определить, являются ли симптомы опасными, для здоровья, требующие профессиональную медицинскую помощь или можно справиться с этим самостоятельно, следуя инструкциям, подготовленными экспертами. Поиск заболеваний происходит на основе симптомов, которые были выявлены. Приложение предоставляет не только список нужных лекарственных средств для каждого заболевания, но и дополнительные рекомендации, которые помогут скорейшему выздоровлению.

Поскольку подбор лекарственных средств происходит на основе экспертных знаний, то для развития базы знаний, также предоставлен специальный инструмент.

Целью данного дипломного проекта является создание такой автоматизированной системы, которая бы смогла:

* предоставить возможность создания базы знаний заболеваний и методы их лечения, доступной любому пользователю;
* экономить время пользователей, упрощая процесс назначения лечения;
* частично разгрузить различные органы здравоохранения.

1 АНАЛИТИЧЕСКИЙ ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ И СРЕДСТВ ПО ВЫБОРУ ЛЕКАРСТВЕННЫХ ПРЕПАРАТОВ

* 1. Анализ предметной области

Лекарственное средство – вещество или смесь веществ [синтетического](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B5%D1%82%D0%B8%D0%BA%D0%B0) или природного происхождения в виде [лекарственной формы](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D0%BA%D0%B0%D1%80%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0) ([таблетки](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B1%D0%BB%D0%B5%D1%82%D0%BA%D0%B8), [капсулы](https://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D0%B0_(%D0%BB%D0%B5%D0%BA%D0%B0%D1%80%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0)), [раствора](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D1%82%D0%B2%D0%BE%D1%80_(%D0%BB%D0%B5%D0%BA%D0%B0%D1%80%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0)), [мази](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%B7%D0%B8) и т. п.), применяемые для [профилактики](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%84%D0%B8%D0%BB%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B0_(%D0%BC%D0%B5%D0%B4%D0%B8%D1%86%D0%B8%D0%BD%D0%B0)), диагностики и [лечения](https://ru.wikipedia.org/wiki/%D0%9B%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [заболеваний](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B1%D0%BE%D0%BB%D0%B5%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Перед употреблением в медицинской практике лекарственные средства должны проходить [клинические исследования](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B8%D0%BD%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D0%B8%D1%81%D1%81%D0%BB%D0%B5%D0%B4%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) и получать разрешение к применению.

Знание механизма действия позволяет осмысленно выбрать необходимый препарат для лечения конкретного заболевания. Кроме того, знание механизма действия необходимо также для правильного комбинирования лекарств и предвидения возможного возникновения нежелательных эффектов.

Знание фармакокинетики лекарственного средства дает врачу возможность осуществить индивидуальный подбор лекарственной терапии данному больному, исходя из особенностей функционирования его организма. Кроме того, знание фармакокинетики препарата позволяет предвидеть появление нежелательных эффектов, а также помогает выбрать оптимальный режим дозирования при данном пути введения для того, чтобы обеспечить терапевтическую концентрацию лекарственного вещества в области рецептора.

Активным средством в руках врача, с помощью которого он может влиять на течение того или иного заболевания, является лекарственный препарат.

Лекарственное средство в руках знающего врача приносит огромную пользу людям. Незнание лекарственных средств, неумение пользоваться ими, низкие морально-этические требования к себе могут привести к непоправимым последствиям для больного человека.

Нередко перед врачом стоит сложная задача – выбрать из большого арсенала лекарственных средств не только самое эффективное, но и наименее токсичное, а также уменьшить риск появления побочного действия. Это в значительной мере обусловлено тем, что при различных условиях одно и то же вещество может оказаться лекарством или ядом. Так, стрихнин, морфин и другие ядовитые и сильнодействующие лекарственные вещества в сравнительно небольших, так называемых терапевтических дозах оказывают лечебный эффект. С увеличением доз этих лекарственных средств выше допустимых они могут проявлять токсическое действие, нередко приводящее к тяжелым последствиям. Иногда обычные дозы лекарственных веществ вместо желаемого действия могут оказать отрицательное влияние на организм, что связывают с индивидуальной чувствительностью больных к этому лекарственному веществу.

Любое лекарственное средство, покупаемое в аптеке, сопровождено специ­альной инструкцией по применению. Но часто ей пренебрегают. Между тем, соблюдение или несоблюдение правил приема может оказать большое, если не решающее, влияние на действие лекарства. Например, при приеме внутрь пища, а также желудочный сок, пищеварительные ферменты, желчь, которые выделяются в процессе ее переваривания, могут взаимодействовать с лекарственными веществами и изменять их свойства. Именно поэтому совсем не безразлично, когда лекарство будет принято: натощак, во время или после еды.

* 1. Обзор аналогов

На данный момент существует несколько web-приложений, которые предоставляют подбор и поиск лекарственных средств. Данные приложения позволяют быстро найти подробную информацию о нужном препарате, а также найти средства их реализаций. В качестве аналога, были рассмотрены следующие web-приложения:

* https://tabletka.by/ - приложение по поиску лекарственных средств в Беларуси;
* https://103.by/ - международная медицинская платформа.

***1.2.1*** Приложение tabletka.by предоставляет услуги, позволяющие пользователю производить поиск лекарственных средств и товаров аптечного ассортимента в данных, полученных от аптек в рамках соглашения между аптеками и приложением tabletka.by. На рисунке 1.1 представлена главная страница сайта tabletka.by.

На главной странице сразу же можно выбрать регион в котором будет выполняться поиск лекарственных средств. Также сразу же доступно поле для ввода наименования лекарственного средства. Данное поле автоматические подгружает список лекарств, подходящий под введенные данные.

Также на главное странице отображается небольшая статистика приложения, а именно: количество доступных аптек и количество выполненных запросов за день.

A screenshot of a computer

Description automatically generated

Рисунок 1.1 – Главная страница приложения tabletka.by

Для удобства пользования сайтом данные по лекарственным средствам, медицинской технике, товарам аптечного ассортимента систематизируются и приводятся к единому написанию (рисунок 1.2). Прочий ассортимент представлен в поиске в написании справочников аптек.

A screenshot of a cell phone

Description automatically generated

Рисунок 1.2 – Данные по лекарственному средству

Также приложение предоставляет данные об аптеках, где можно приобрести данный препарат. На рисунке 1.3 представлена страница со списком аптек. Для каждой аптеки указан адрес, телефон, график работы и цена препарата. Есть возможность забронировать препарат на определенный промежуток времени. Также есть возможность просмотреть местоположение аптеки на карте (рисунок 1.4).

A screenshot of a cell phone

Description automatically generated

Рисунок 1.3 – Список аптек реализующие продажу необходимого препарата

A screenshot of a computer

Description automatically generated

Рисунок 1.4 – Местоположение аптеки на карте

Любая аптека может подключиться к данному сервису. Для подключения к сервису требуется наличие программы предметно-количественного учета с возможностью выгрузки отчета по остаткам и подключение к сети интернет. Подробная информация о формате выгрузки и реквизитам подключения определяется разработчиками сервиса.

***1.2.2*** Международная медицинская платформа 103 – проект компании ARTOX, специализирующейся на тематических агрегаторах. В 2013 году появился 103.BY – белорусский портал и мобильное приложение с максимально полным каталогом медицинских учреждений, сервисом поиска лекарств в аптеках и полезным контентом о здоровье и красоте. Главная страница сайта 103.by представлена на рисунке 1.5.

A screenshot of a social media post

Description automatically generated

Рисунок 1.5 – Главная страница приложения 103

К сервису поиска лекарств подключены более 3 600 аптек по всей Беларуси, и их количество постоянно растет. Любой пользователь легко может найти нужное лекарство по минимальной цене в выбранном регионе. На рисунке 1.6 представлен пример поиска лекарственных средств.

В форме для поиска аналогично присутствует поле для выбора региона и текстовое поля для ввода наименования, которое автоматически подгружает список подходящих лекарственных средств.

A screenshot of a cell phone

Description automatically generated

Рисунок 1.6 – Список найденных лекарственных средств

Также аптеки могут быть отсортированы по близости к текущему местонахождению пользователя и отображаться на карте (рисунок 1.7). Доступны функции онлайн-бронирования, поиска наборов лекарств (можно выбрать сразу несколько препаратов и найти, где полный набор есть в наличии по минимальной цене), инструкции к препаратам, аналоги лекарств.

A close up of a map

Description automatically generated

Рисунок 1.7 – Список аптек реализующие необходимый препарат

Каждое учреждение имеет персональную страницу на отдельном поддомене, которая фактически является сайтом учреждения и содержит всю необходимую для пользователей информацию.

Так, по каждому медицинскому учреждению будет как минимум справочная информация: адрес, контакты, время работы, отзывы. Для расширенного формата доступен полный функционал: сведения об организации, перечень услуг, их описание и цены, фото- и видеогалереи, данные о специалистах, новости, акции и скидки.

* 1. Требования к проектируемому программному обеспечению

На основании состояния предметной области, анализа используемых в ней методах и инструментах, было сформированы требования к разрабатываемой системе

Цель разработки: разработать кроссплатформенное веб-приложение по выбору лекарственных препаратов на основе экспертных знаний.

Требуемый функционал:

* регистрация, авторизация пользователей;
* система ролей;
* создание заболевания, с необходимым описанием, добавлением лечения и добавлением рекомендаций;
* поиск заболеваний по определенным фильтрам;
* создание необходимых рекомендация;
* создание симптомов;
* создание отзывов пользователями;
* анализ отзывов, для каждого заболевания.

В приложении присутствует система ролей. Для всех ролей создано единое браузерное приложение с применением технологии Lazy Loading, что позволит не создавать для каждой роли отдельное приложение и уменьшить время загрузки приложения.

Приложение должно иметь удобный и понятный интерфейс для обычного пользователя. Приложение должно работать во всех современных браузерах.

Список используемых технологий и их обоснование представлены в разделе 2.5 (Выбор средств разработки). Таким образом, в данном разделе выполнен анализ предметной области и подготовлено общее техническое задание.

1. Архитектура программного обеспечения
   1. **Описание построения веб-приложения**

В техническом задании было указано, что программный комплекс будет разрабатываться как клиент-серверное приложение.

Клиент-сервер – это, сетевая архитектура, в которой сетевая нагрузка распределена между сервером и клиентом. Обычно они расположены на разных вычислительных машинах и взаимодействуют между собой посредством сетевых протоколов, но также могут быть расположены и на одной вычислительной машине. Схема клиент-серверного приложения представлена на рисунке 2.1.

Преимущества такой системы:

* отсутствие дублирования кода программы-сервера программами-клиентами;
* так как все вычисления выполняются на сервере, то требования к компьютерам, на которых установлен клиент, снижаются;
* все данные хранятся на сервере, который, как правило, защищен гораздо лучше большинства клиентов;
* на сервере проще организовать контроль полномочий, чтобы разрешать доступ к данным только клиентам с соответствующими правами доступа.



Рисунок 2.1 – Схема клиент-серверного приложения

Однако у такой системы существуют и недостатки. Например, недоступность сервера может сделать неработоспособной все вычислительную сеть.

Серверное приложение разработано с применением архитектура REST. REST изначально был задуман как простой и однозначный интерфейс для управления данными, предполагавший всего несколько базовых операций с непосредственным сетевым хранилищем (сервером): извлечение данных (GET), сохранение (POST), изменение (PUT/PATCH) и удаление (DELETE). Разумеется, этот перечень всегда сопровождался такими опциями, как обработка ошибок в запросе, разграничение доступа к данным и валидация входящих данных, в общем, всеми возможными проверками, которые сервер выполняет перед тем, как выполнить желание клиента.

Помимо этого REST имеет ряд архитектурных принципов.

Независимость сервера от клиента – серверы и клиенты могут быть мгновенно заменены другими независимо друг от друга, так как интерфейс между ними не меняется. Сервер не хранит состояний клиента.  
Уникальность адресов ресурсов – каждая единица данных имеет свой собственный уникальный URL, который, по сути, целиком является однозначным идентификатором ресурса.

Независимость формата хранения данных от формата их передачи – сервер может поддерживать несколько различных форматов для передачи одних и тех же данных (JSON, XML и т.д.), но хранит данные в своем внутреннем формате, независимо от поддерживаемых.

Присутствие в ответе всех необходимых метаданных – помимо самих данных сервер должен возвращать детали обработки запроса, например, сообщения об ошибках, различные свойства ресурса, необходимые для дальнейшей работы с ним, например, общее число записей в коллекции для правильного отображения постраничной навигации.

Серверное приложение разработано с применением паттерна «Репозиторий» в уровне доступа к данным, который считается одним из наиболее часто используемых паттернов при работе с данными. «Репозиторий» позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой. Данный паттерн добавляет программе гибкость при работе с разными типами подключений.

База данных разрабатывалась с использованием ORM Sequelize [2], которая позволяет, при отсутствии базы данных, создать ее на основе миграций и заполнить базу данных начальными данными при помощи сидов.

* 1. **Описание подхода построения SPA**

В качестве клиентского приложения было разработано Single Page Application. SPA –  это [веб-приложение](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D0%BF%D1%80%D0%B8%D0%BB%D0%BE%D0%B6%D0%B5%D0%BD%D0%B8%D0%B5), использующее единственный [HTML-документ](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) как оболочку для всех веб-страниц и организующий взаимодействие с пользователем через динамически подгружаемые [HTML](https://ru.wikipedia.org/wiki/HTML), [CSS](https://ru.wikipedia.org/wiki/CSS), [JavaScript](https://ru.wikipedia.org/wiki/JavaScript" \o "JavaScript), обычно посредством [AJAX](https://ru.wikipedia.org/wiki/AJAX).

Как и каждая технология SPA имеет как свои плюсы так и минусы.

SPA приложение отлично работают на устройствах как стационарных, так и мобильных. Персональные компьютеры, планшеты, смартфоны, и, в конце концов, простые телефоны могут беспрепятственно работать с сайтами построенных по принципу SPA. Работа на большом количестве устройств, а значит, создав одно приложение, приложение получает гораздо большую аудиторию пользователей нежели при использовании стандартного подхода.

Богатый пользовательский интерфейс, так называемый User Experience. Так как web-страница одна, построить богатый, насыщенный пользовательский интерфейс гораздо проще. Проще хранить информацию о сеансе, управлять состояниями представлений и управлять анимацией.

SPA существенно сокращает количество повторных загрузок страниц. Если сайт использует шаблон, то вместе с основным содержанием какой-либо страницы посетитель сайта обязательно загружает разметку шаблона.

Составляющие SPA:

* SPA поддерживает клиентскую навигации, все переходы пользователя по модулям-страницам однозначно фиксируются в истории навигации, то есть если пользователь скопирует и откроет ссылку на внутреннюю модуль-страницу в другом браузере или окне, он попадет на соответствующую страницу;
* SPA размещается на одной web-странице, значит все необходимое для работы сайта скрипты и стили должны быть определены в одном месте проекта – на единственной web-странице;
* SPA хранит постоянно состояние работы клиента в кэше браузера или в Web Storage;
* SPA загружает все скрипты требующиеся для старта приложения при инициализации web-страницы;
* SPA постепенно подгружает модули по требованию.
  1. **Описание паттерна Promise**

Promise – это шаблон проектирования для организации кода при работе с асинхронными операциями.

Шаблон promise используется для операций, вычисление которых занимает неопределенное время. Примером подобной операции может быть сетевой запрос, когда приложение запрашивает данные у API и не можем точно определить, когда будет получен ответ.

Promise предоставляет стандартизированный и понятный метод решения асинхронных задач, который должен обрабатывать результат последовательно.

В JavaScript promise – это специальный объект, который содержит свое состояние [3]. Вначале pending (ожидание), затем – одно из: fulfilled (выполнено успешно) или rejected (выполнено с ошибкой). На рисунке 2.2 представлена схема работы шаблона promise.

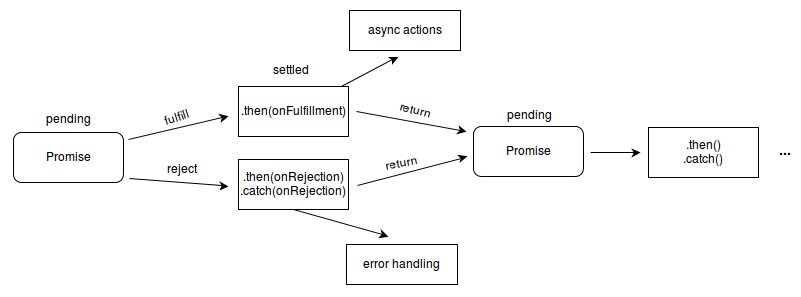


Рисунок 2.2 – Схема работы шаблона promise

К шаблону можно добавлять функции обратного вызова двух типов:

* onFulfilled – срабатывают, когда promise в состоянии «выполнен успешно»;
* onRejected – срабатывают, когда promise в состоянии «выполнен с ошибкой».

Способ использования такой:

* код, которому надо сделать что-то асинхронно, создает объект promise и возвращает его;
* внешний код, получив promise, навешивает на него обработчики;
* по завершении процесса асинхронный код переводит promise в состояние fulfilled (с результатом) или rejected (с ошибкой), при этом автоматически вызываются соответствующие обработчики во внешнем коде.

**2.4 Node.JS платформа для выполнения JavaScript на серверной стороне**

NodeJS – это серверная платформа, основанная на движке V8 (разрабатывается компанией Google), который транслирует язык JavaScript в машинный код. Также в платформу входят дополнительные модули, написанные на C++ для работы с потоками ввода-вывода, файловой системой и другими системными API. Данная платформа предназначена для разработки производительных и масштабируемых веб-приложений [1].

Основным языком программирования для данной платформы является JavaScript и поддерживаются самые актуальные стандарты языка, с некоторыми исключениями. Для языка существует множество библиотек и все они расположены в пакетном менеджере npm или yarn.

В NodeJS помимо JavaScript можно использовать встроенные модули, написанные на языке C++.

NodeJS работает в асинхронно-событийной модели. Все API-интерфейсы библиотеки Node.js не блокируют загрузку. NodeJS является однопоточным приложением.

NodeJS представляет себя в качестве технологического решения при построении приложений, связанные с вводом-выводом данных, для приложений потоковой передачи данных, приложений обрабатывающих большой объем данных в режиме реального времени (DIRT), приложениях на основе API JSON.

В сравнении с другими платформами для разработки веб приложений NodeJS выигрывает в производительности и количестве одновременно обрабатываемых запросов.

* 1. Выбор средств разработки

Весь проект можно разделить на четыре основные части, а именно:

* создание браузерного приложения;
* создания REST API приложения, для отправки и обработки данных, а также для взаимодействия с базой данных;
* создание сервиса, для интеграции со сторонними сервисами, предоставляющие базу лекарственных средств;
* проектирование и создание базы данных.

Для создания браузерного приложения были применены следующие технологии: React, Redux, SCSS, Typescript, Webpack, Lodash, Babel.

Для создания REST API приложения и сервиса для интеграции были применены технологии: Node.js, Typescript, Express, Sequelize.

Для хранения данных была применена база данных MySql. Для удобного управления базой данных было использовано специальное средство администрирования Workbench.

При создании проекта, использовалась среда разработки JetBrains WebStorm.

React – это библиотека JavaScript, которая используется для создания пользовательского интерфейса [4]. React был создан компанией Facebook. Данная библиотека позволяет создавать одностраничные приложения. Также особенностью данной библиотеки является, использование Virtual Document Object Model. Он создает кеш структуру страницы в памяти, с ее помощью находит разницу между прошлым и текущем состоянием интерфейса для оптимизации обновления Document Object Model. Таким образом страница обновляется не полностью, а только некоторая ее часть, за счет чего увеличивается скорость работы приложения.

Сегодняшний мир веб-сайтов трудно представить без языка JavaScript [3]. Это то, что делает живыми веб-страницы, которые мы каждый день просматриваем в своем веб-браузере. Первоначально JavaScript обладал довольно небольшими возможностями. Его цель состояла лишь в том, чтобы добавить немного поведения на веб-страницу.

Однако развитие веб-среды, появление HTML5 и технологии Node.js открыло перед JavaScript гораздо большие горизонты. Сейчас JavaScript продолжает использоваться для создания веб-сайтов, только теперь он предоставляет гораздо больше возможностей. Также он применяется как язык серверной стороны. JavaScript является языком с динамической типизацией, что дает определенные плюсы и минусы. В больших проектах отсутствие статической типизации ведет к большему количеству багов. Для решения данной проблемы была разработана надстройка TypeScript.

TypeScript – это строго типизированный и компилируемый язык представленный на основе JavaScript. Хотя на выходе компилятор создает все тот же JavaScript, который затем исполняется браузером. Однако строгая типизация уменьшает количество потенциальных ошибок, которые могли бы возникнуть при разработке на JavaScript. TypeScript реализует многие концепции, которые свойственны объектно-ориентированным языкам, как, например, наследование, полиморфизм, инкапсуляция и модификаторы доступа и так далее.

Часто React приложение используется с Redux. Redux – библиотека для JavaScript с открытым исходным кодом, предназначенная для управления состоянием приложения. Данная библиотека реализует архитектуру единого хранилища данных в приложении.

Препроцессор SCSS позволяет использовать функции недоступные в самом CSS, например, переменные, вложенности, функции, наследование и другие приятные вещи.

Node.js представляет среду выполнения кода на JavaScript, которая построена на основе движка JavaScript Chrome V8, который позволяет транслировать вызовы на языке JavaScript в машинный код. Node.js прежде всего предназначен для создания серверных приложений на языке JavaScript.

Библиотека Express значительно упрощает создание REST API приложения на Node.js. Она упрощает работу с маршрутами, отправкой статических файлов, а также упрощает работу с запросами.

ORM – это технология программирования, который связывает базу данных с моделью объектно-ориентированных языков программирования. Он создает для СУБД виртуальный объектный интерфейс. Библиотека Sequelize – это Node.js ORM система для работы с такими базами данных, как PostgreSQL, MySQL, SQLite и MSSQL [2].

  Webpack представляет популярный упаковщик модулей, который позволяет автоматизировать создание из нескольких файлов один.

Babel – транспайлер, переписывающий код на ES-NEXT в код на предыдущем стандарте.

* 1. **Экспертные системы**

Экспертные системы это направление исследований в области искусственного интеллекта по созданию вычислительных систем, умеющих принимать решения, схожие с решениями экспертов в заданной предметной области.

Как правило, экспертные системы создаются для решения практических задач в некоторых узкоспециализированных областях, где большую роль играют знания «бывалых» специалистов. Экспертные системы были первыми разработками, которые смогли привлечь большое внимание к результатам исследований в области искусственного интеллекта.

Экспертные системы имеют одно большое отличие от других систем искусственного интеллекта: они не предназначены для решения каких-то универсальных задач, как например [нейронные сети](http://www.aiportal.ru/articles/neural-networks/neural-networks.html) или [генетические алгоритмы](http://www.aiportal.ru/articles/genetic-algorithms/genetic-algorithms.html). Экспертные системы предназначены для качественного решения задач в определенной разработчиками области, в редких случаях – областях.

Экспертное знание – это сочетание теоретического понимания проблемы и практических навыков ее решения, эффективность которых доказана в результате практической деятельности экспертов в данной области. Фундаментом экспертной системы любого типа является база знаний, которая составляется на основе экспертных знаний специалистов. Правильно выбранный эксперт и удачная формализация его знаний позволяет наделить экспертную систему уникальными и ценными знаниями. Врач, к примеру, хорошо диагностирует болезни и эффективно назначает лечение, не потому, что он обладает некими врожденными способностями, а потому что имеет качественное медицинское образование и большой опыт в лечении своих пациентов. Поэтому ценность всей экспертной системы как законченного продукта на 90% определяется качеством созданной базы знаний.

Экспертная система – это не простая программа, которая пишется одним или несколькими программистами. Экспертная система является плодом совместной работы экспертов в данной предметной области, инженеров по знаниям и программистов. Но стоит отметить, что встречаются случаи, когда программы пишутся самими экспертами в данной области.

Эксперт предоставляет необходимые знания о тщательно отобранных примерах проблем и путей их решения. Например, при создании экспертной системы диагностики заболеваний врач рассказывает инженеру по знаниям об известных ему заболеваниях. Далее эксперт раскрывает список симптомов, которые сопровождают каждое заболевание и в заключение рассказывает об известных ему методах лечения. Инженер по знаниям, формализует всю полученную информацию в виде базы знаний и помогает программисту в написании экспертной системы.

На сегодняшний день создано уже большое количество экспертных систем. С помощью них решается широкий круг задач, но исключительно в узкоспециализированных предметных областях. Как правило, эти области хорошо изучены и располагают более менее четкими стратегиями принятия решений. Сейчас развитие экспертных систем несколько приостановилось, и этому есть ряд причин.

Передача экспертным системам «глубоких» знаний о предметной области является большой проблемой. Как правило, это является следствием сложности формализации эвристических знаний экспертов.

Экспертные системы неспособны предоставить осмысленные объяснения своих рассуждений, как это делает человек. Как правило, экспертные системы всего лишь описывают последовательность шагов, предпринятых в процессе поиска решения.

Отладка и тестирование любой компьютерной программы является достаточно трудоемким делом, но проверять экспертные системы особенно тяжело. Это является серьезной проблемой, поскольку экспертные системы применяются в таких критичных областях, как управление воздушным и железнодорожным движением, системами оружия и в ядерной промышленности.

Экспертные системы обладают еще одним большим недостатком: они неспособны к самообучению. Для того, чтобы поддерживать экспертные системы в актуальном состоянии необходимо постоянное вмешательство в базу знаний инженеров по знаниям. Экспертные системы, лишенные поддержки со стороны разработчиков, быстро теряют свою востребованность.

В заключение стоит отметить, что несмотря на все эти ограничения и недостатки, экспертные системы уже доказали всю свою ценность и значимость во многих важных приложениях.

2.7 Функциональная модель программного комплекса

При реализации программы следует учесть разбиение функционала в зависимости от роли пользователя и дать каждому доступ к определенным функциям и ограничить доступ к иным.

Данной ограничение реализовано при помощи промежуточных обработчиков. Программа предусматривает разделение на 2 роли, а именно на эксперта и обычного пользователя. Каждый из пользователей должен быть зарегистрирован, чтобы иметь возможность использовать приложение. Если это новый пользователь, то он может зарегистрироваться самостоятельно.

Авторизованный пользователь имеет следующий функционал:

* поиск заболеваний по определенному фильтру;
* просмотр подробной информации о заболевании;
* создание отзыва о каждом заболевании.

Эксперт имеет следующий функционал:

* создание, редактирование, удаление заболевания;
* создание, редактирование, удаление рекомендаций;
* создание, редактирование, удаление симптомов;
* просмотр и анализ отзывов.

Любые (в том числе и программные) системы проектируются с учетом того, что в процессе своей работы они будут использоваться людьми и/или взаимодействовать с другими системами. Поэтому основным и самым распространенным видом uml-диаграмм являются диаграммы вариантов использования. Данная диаграмма представлена на рисунке 2.3.

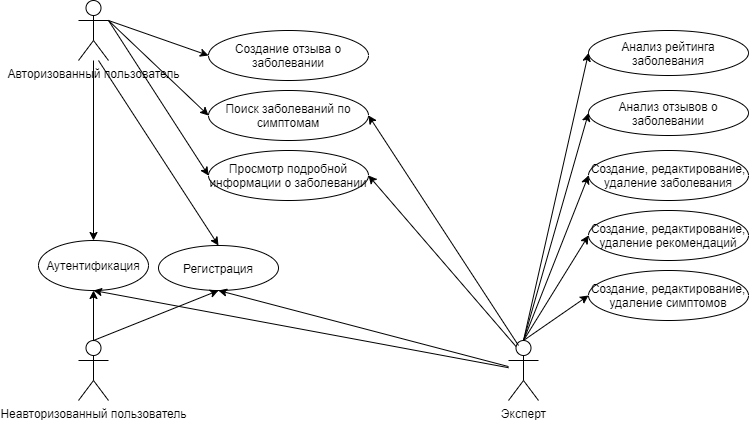


Рисунок 2.3 – Диаграмма вариантов использования

2.8 Архитектура системы

Под архитектурой программных систем будем понимать совокупность решений относительно:

* организации программной системы;
* выбора структурных элементов, составляющих систему и их интерфейсов;
* поведения этих элементов во взаимодействии с другими элементами;
* объединение этих элементов в подсистемы;
* архитектурного стиля, определяющего логическую и физическую организацию системы: статические и динамические элементы, их интерфейсы и способы их объединения.

Архитектура программной системы охватывает не только ее структурные и поведенческие аспекты, но и правила ее использования и интеграции с другими системами, функциональность, производительность, гибкость, надежность, возможность повторного применения, полноту, экономические и технологические ограничения, а также вопрос пользовательского интерфейса. Архитектура разрабатываемой системы представлена на рисунке 2.4.

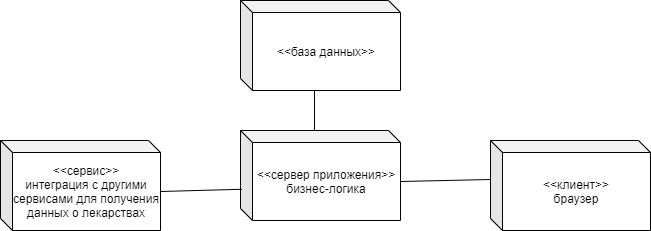


Рисунок 2.4 – Диаграмма архитектуры системы

Для разрабатываемого веб-приложения была выбрана классическая клиент-серверная архитектура, т.е. состоящая из сервера, узлов(сервисов) и клиентов.

Для разгрузки серверной части было принято решение отделить интеграцию со сторонними сервисами в отдельный сервис.

Так как в организациях уже развернута и налажена собственная сеть, внедрение системы с такой архитектурой не приведет к проблемам и материальным затратам.



2.9 Информационная модель

Основу автоматизированной системы составляют базы данных. Широкое распространение получили реляционные (табличные) базы данных. Для безотказного пользования базой данных необходимо тщательное ее проектирование, состоящее в основном в определении состава таблиц и связей между ними.

Основные задачи проектирования БД:

* обеспечение надежного хранения в базе всей необходимой информации;
* обеспечение возможности получения данных по всем необходимым запросам;
* исключение избыточности и дублирования данных;
* обеспечение целостности базы данных.

В процессе проектирования базы данных для разрабатываемого программного комплекса были выполнены следующие этапы:

* уточнение задач: выявление задач системы, требующих коммуникаций с БД, выделение их в группы;
* анализ данных: составление подробного перечня всех данных, необходимых для решения каждой задачи;
* определение структуры данных: упорядочивание данных по объектам и определение связей между ними, или нормализация;
* тестирование в результате выполнения запросов к базе данных и ее совершенствование.

Последний этап может проводится в процессе разработки приложения.

Физическая модель базы данных определяет способ размещения данных в среде хранения и способы доступа к этим данным, которые поддерживаются на физическом уровне.

Схема используемой базы данных представлена на рисунке 2.5.

В таблице 2.1. представлена схема сущности «Заболевание». Данная сущность хранит всю информацию о заболеваниях.

Таблица 2.1 – Схема таблицы заболевания (Illnesses)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| name | VARCHAR | Название заболевания |
| description | VARCHAR | Подробное описание заболевания |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

Сущность «Лекарство» – хранит информацию о лекарствах, которые были получены со сторонних сервисов. Схема сущности представлена в таблице 2.2.

Таблица 2.2 – Схема таблицы лекарства (Medicines)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| name | VARCHAR | Название лекарства |
| description | TEXT | Подробное описание лекарства |
| wayUse | TEXT | Инструкция по применению лекарства |
| typeMedicineId | INT | Идентификатор типа лекарства |
| routeId | INT | Идентификатор пути ввода лекарства |
| productionMedicineMethodId | INT | Идентификатор порядка отпуска лекарства |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

Сущность «Группа» – хранит информацию о лекарственных группах. Каждое лекарственное средство принадлежит к определенной группе. В каждой группе может находиться неограниченное количество лекарственных средств. Схема сущности представлена в таблице 2.3.

Таблица 2.3 – Схема таблицы группы (Groups)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| name | VARCHAR | Наименование группы |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

Сущность «Порядок отпуска» – хранит информацию о всех доступных видах отпуска лекарственных средств. Схема сущности представлена в таблице 2.4.

Таблица 2.4 – Схема таблицы порядок отпусков (ProductionMedicineMethods)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| name | VARCHAR | Наименование порядка отпуска |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

Сущность «Рекомендация» – хранит информацию о рекомендациях, которые могут быть применены к любому из заболеваний. Схема сущности представлена в таблице 2.5.

Таблица 2.5 – Схема таблицы рекомендации (Recommendations)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| name | VARCHAR | Наименование рекомендации |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

Сущность «Путь введения» – хранит информацию о методах введения или приема лекарственного средства. Схема сущности представлена в таблице 2.6.

Таблица 2.6 – Схема таблицы пути ввода (Routes)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| name | VARCHAR | Наименование пути ввода |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

Сущность «Симптом» – хранит информацию о различных симптомах, которые характерны для различных заболеваний. Схема сущности представлена в таблице 2.7.

Таблица 2.7 – Схема таблицы симптомы (Symptoms)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| name | VARCHAR | Наименование симптома |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

Сущность «Тип лекарства» – хранит информацию о типах лекарственных средств. У каждого лекарственного средства есть определенный тип. Схема сущности представлена в таблице 2.8.

Таблица 2.8 – Схема таблицы типы лекарств (Typemedicines)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| name | VARCHAR | Наименование типа лекарства |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

Сущность «Пользователь» – хранит информацию о пользователях. Схема сущности представлена в таблице 2.9.

Таблица 2.9 – Схема таблицы пользователи (Users)

|  |  |  |
| --- | --- | --- |
| Название поля | Тип данных | Описание |
| 1 | 2 | 3 |
| id | INT | Целочисленный идентификатор, значение устанавливается при помощи автоматического прибавления единицы к предыдущему значению |
| firstname | VARCHAR | Имя пользователя |
| lastname | VARCHAR | Фамилия пользователя |
| email | VARCHAR | Электронная почта пользователя |
| password | VARCHAR | Пароль пользователя |
| createdAt | DATETIME | Дата создания записи |
| updatedAt | DATETIME | Дата последнего обновления записи |

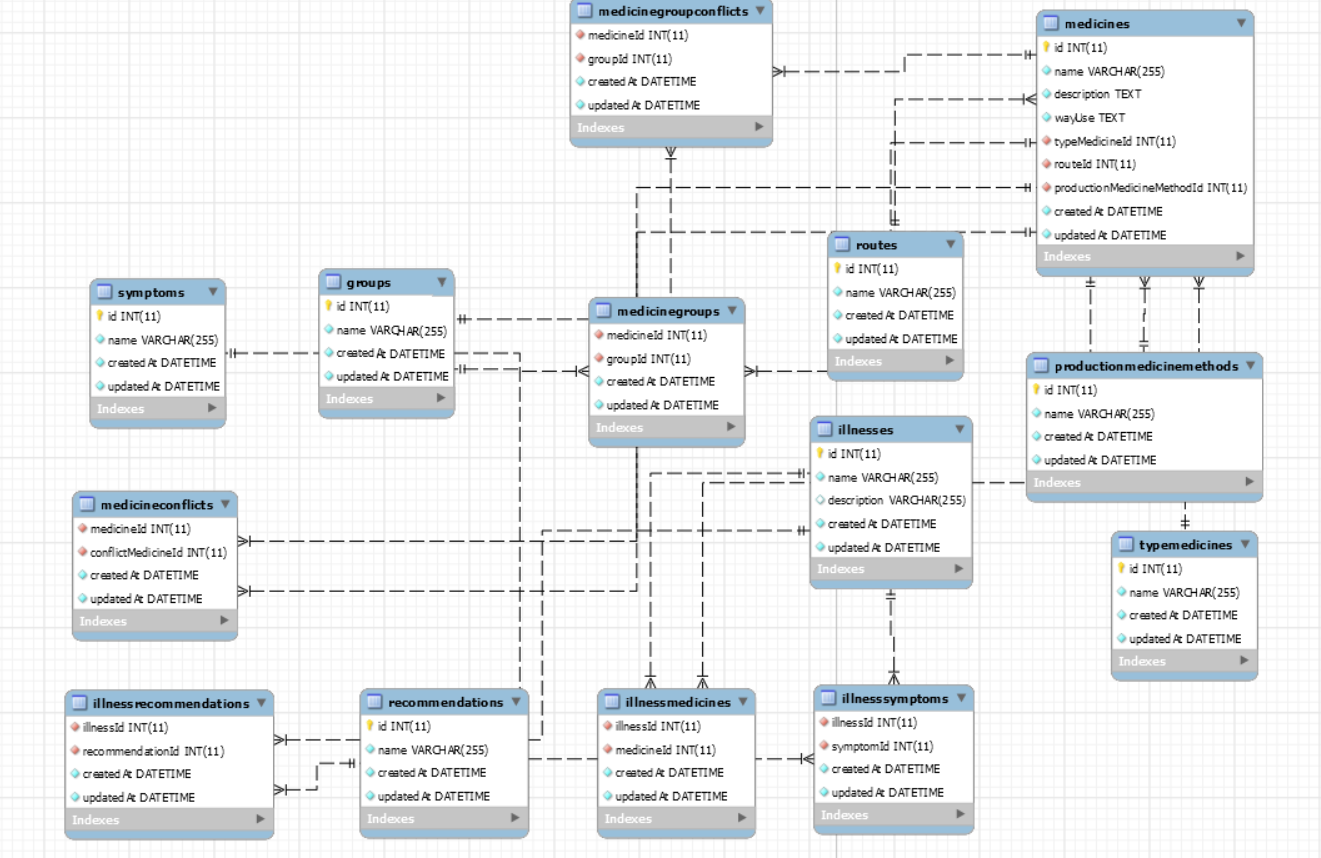


Рисунок 2.5 – ER-диаграмма базы данных

# **3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ**

**3.1 Настройка рабочего окружения и создание проекта**

Прежде чем начать разработку приложения, необходимо установить и настроить соответствующие инструменты.

В первую очередь был создан GIT репозиторий, что позволяет удобно ввести поэтапную разработку и где будет находиться вся история разработки проекта. Также данный репозиторий был размещен на удаленный сервер, для быстрого доступа к проекту на различных устройствах.

Для запуска сервера необходимо установить среду исполнения Node.js и один из пакетных менеджеров. В данном случае используется пакетный менеджер yarn, для более быстрой установки пакетов. Среда исполнения Node.js используется, как для запуска сервера, так и для запуска сборщика React приложения.

В данном приложении используется язык программирования TypeScript, но поскольку среда исполнения Node.js не поддерживает данный язык, то сначала нужно установить специальный компилятор, который будет транслировать TypeScript код в JavaScript код и только потом запускать его в данной среде исполнения.

Для сборки React приложения применяется сборщик Webpack. Данный сборщик позволяет собрать весь проект, включающий в себя огромное количество файлов, компонентов, а также стилей, в один JavaScript файл. Также данный сборщик позволяет использовать язык TypeScript и в React приложении, а также препроцессор SASS.

Поскольку каждый разработчик имеет свой стиль написания кода, в процессе разработки используется пакет EsLint. Данный пакет проверяет стиль написания кода, в соответствии с заданными конфигурациями, перед каждым фиксированием кода, что позволяет избежать различного стиля кода.

Для ведения истории в едином формате, используется пакет commitlint, который перед каждым фиксированием кода проверяет его формат, в соответствии с конфигурациями.

Данные инструменты помогают избежать попадания на удаленный репозиторий, кода и истории, которые не соответствуют требованиям.

**3.2 Структура и способ построения приложения**

**3.2.1** Сервер состоит из множества слоев, выполняющие определенные задачи. На рисунке 3.1 представлена структура серверного приложения. Каждая директория отвечает за определенный слой в приложении. Каждый слой приложения независим, что позволяет с легкостью заменить или переделать определенную часть приложения. Из основного можно выделить следующие директории.

В директории app находится основная логика приложения, а именно: проверка данных на валидность, проверка приходящих данных на соответствие заданной модели, основные алгоритмы.

В директория config находятся все настройки приложения по умолчанию, чтобы подменить данные настройки используется специальный текстовый файл .env. Использование данного файла позволяет изменять настройки приложения без изменения самого приложения и без повторной сборки.

Директория domain предназначена для всех моделей в приложении. Модели представлены в виде интерфейсов. Для расширения определенных моделей применяется механизм наследования для интерфейсов.

В директории infra находится основная логика для работы с данными. Для быстрого создания готовой базы данных с исходными данными, используются миграции и сиды. Для связи основной логики приложения и логики для работы с данными, используется паттерн репозиторий. Данный паттерн позволяет полностью отделить логику работы с базой данных и основную логику. Также этот паттерн полезен, если мы используем несколько источников данных. Таким образом при смене источника данных, нам всего лишь нужно будет изменить один файл, но при этом он должен обязательно реализовать интерфейс прошлого. Также в директории utils находятся простые, чистые функции, которые используются во всем приложении.

Один из самых важных слоев приложения является его интерфейс. Интерфейсы приложения находятся в директории interfaces. Интерфейсы определяют, каким образом мы можем взаимодействовать с данным приложением. В данном случае используется один интерфейс, а именно: протокол HTTP. При помощи данного интерфейса происходит общение между клиентским приложением и сервера. Поскольку вся логика разбита на множество независимых слоев, для приложения мы можем добавлять неограниченное количество необходимых интерфейсов.

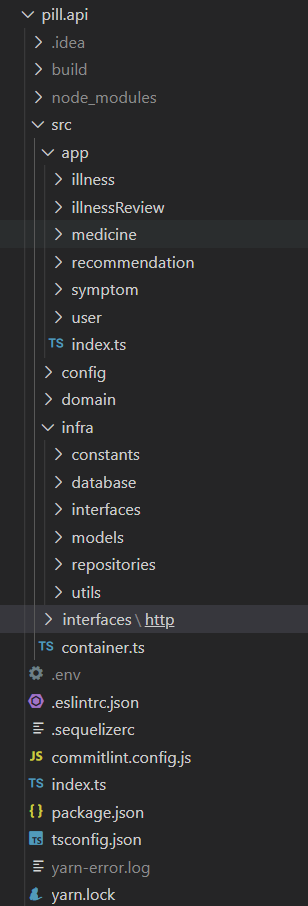
****

Рисунок 3.1 – Структура серверного приложения

**3.2.2** React приложение построено на компонентах [4]. Компоненты представляют основные строительные блоки приложения. Архитектура основывается на взаимосвязанных деревьях компонентов с их собственными интерфейсами ввода/вывода.

Существует два типа компонентов. Первый тип компонента называется сложным. Данные компоненты являются корневыми для каждой страницы. Также они не имеют своей разметки. В этих компонентах находится только основная логика приложения или страницы. Второй тип - это простые компоненты. Данные компоненты не несут никакой логики и предназначены только для отображения данных. Простые компоненты также могут быть вложены в сложные или простые компоненты.

В директории components находятся все компоненты приложения. Все модели приложения находятся в директории models. Директория config предназначена для настройки всего приложения.

Также одна из главных частей приложения является его сборка. Сборка React приложения происходит при помощи сборщика Webpack. Данный сборщик объединяет все используемые файлы в проекте в один. Для настройки сборщика используется специальный файл webpack.config.js.

Поскольку приложение должно поддерживаться во всех популярных браузерах, нужно привести JavaScript код к стандарту который поддерживается всеми необходимыми браузерами. Для этого используется специальный пакет Babel. Он переводит весь JavaScript код с последних стандартов на более ранние, что позволяет постоянно использовать последний новинки языка. Для настройки данного пакета используется файл .babelrc.json.

Важной частью клиентского приложения является реализация единого хранилища данных. Настройка данного хранилища происходит в директории store.

Для реализации единого хранилища данных применяется библиотека Redux. Схема работы данного хранилища представлена на рисунке 3.3. Главной особенностью единого хранилища является доступ к данным в любом месте приложения.

Хранилище представляет собой обычный объект с вложенностями. Основная логика работы хранилища заключается в том, что после какого-либо события компоненты вызывают actions. Каждый action обрабатываются в reducer и после этого обновляется состояние хранилища (store). После каждого обновления хранилища будет происходить обновление страницы.

Структура приложения представлена на рисунке 3.2.

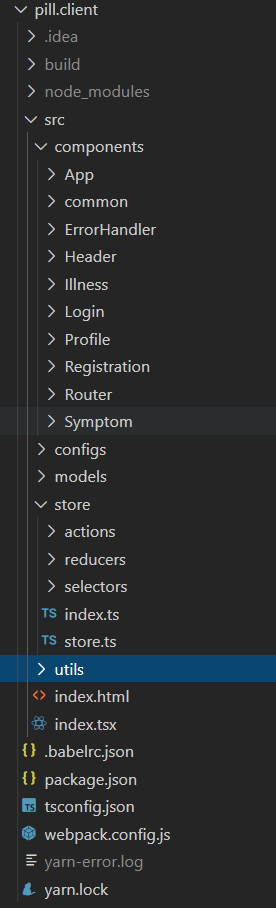
****

Рисунок 3.2 – Структура клиентского приложения

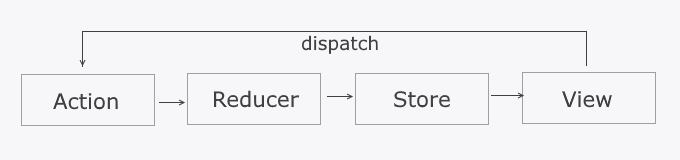


Рисунок 3.3 – Схема работы единого хранилища

**3.3 Алгоритмическое обеспечение**

Основной алгоритмической задачей было расчет рейтинга заболевания, на протяжении конкретного промежутка времени.

Поскольку подбор лекарственных средств осуществляется на основе экспертных знаний, то для анализа данных необходим определенный механизм. Для анализа, была создана система рейтингов для каждого заболевания. Любой пользователь может оставить отзыв о заболевании. Пользователь может поставить оценку каждому разделу, а также конкретному лекарству. На основе введенных данных пользователями, эксперты могут принимать соответствующие решения. Например, заменить лекарственный препарат на другой, более эффективный.

Для расчета рейтинга определенного заболевания применяется формула 3.1.

, (3.1)

где R – рейтинг заболевания;

n – количество отзывов;

k – оценка за конкретный раздел;

h – количество лекарственных средств;

t – оценка конкретного лекарственного средства.

Рейтинг на протяжении всего времени отображается на специальном графике. На рисунке 3.4, можно увидеть изменение рейтинга за каждый день. Рейтинг за каждый день также рассчитывается по формуле 3.1, но при этом показатель n будет зависеть от количества отзывов до определенной даты.

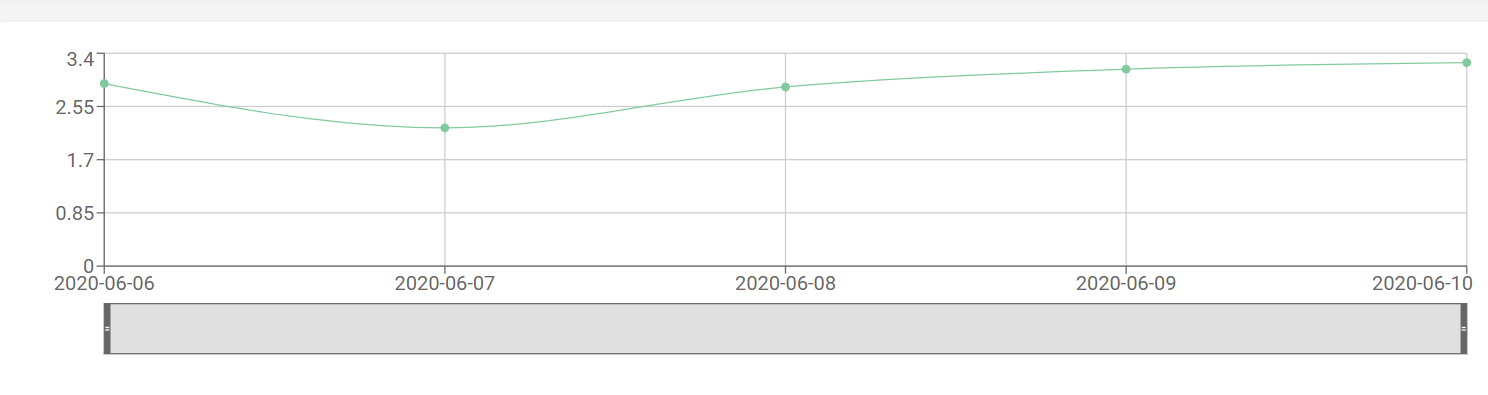


Рисунок 3.4 – График рейтинга заболевания

Таким образом на основе этой системы, эксперты могут анализировать данные и усовершенствовать базу знаний.

**3.4 Интерфейс приложения**

Структура сайта – это иерархия всех страниц сайта, их принадлежность к тем или иным каталогам. С точки зрения пользователя, структура сайта – это навигация, путь, который он должен пройти чтобы попасть на необходимую ему страницу. В первую очередь при разработке структуры сайта следует учитывать интересы пользователя, а потом уже подстраиваться под поисковые системы.

Для удобной навигации по всему приложению, использует навигационное меню с определенными разделами, рисунок 3.5. В навигационном меню содержаться ссылки на следующие страницы: Search illnesses, Create illness, Manage illness, Manage symptoms.

****

Рисунок 3.5 – Навигационное меню приложения

Разделы меню также имеют подразделы, для удобной компоновки навигационных ссылок на различные страницы приложения. Подразделы представлены на рисунке 3.6.

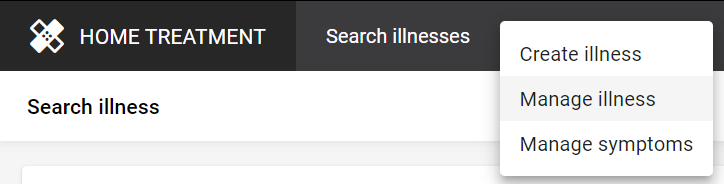


Рисунок 3.6 – Подразделы навигационного меню

Поскольку приложение имеет систему ролей, то для каждой роли доступны определенные страницы.

Авторизованный пользователь имеет доступ к страницам:

* поиск лекарственных средств;
* просмотр подробной информации о лекарстве;
* создание отзыва.

Эксперты имеют доступ к страницам обычного пользователя, а также:

* создание заболевания;
* редактирование заболевания;
* управление заболеваниями;
* анализ заболевания;
* управление симптомами.

Как только пользователь открывает приложение, первая страница, которая появится, будет страница авторизации (рисунок 3.7). Для того чтобы получить доступ к приложению, пользователю необходимо ввести свою уникальную электронную почту и пароль. После заполнения данных полей, пользователь должен нажать кнопку «Login» или, для большего удобства, нажать кнопку «Enter».

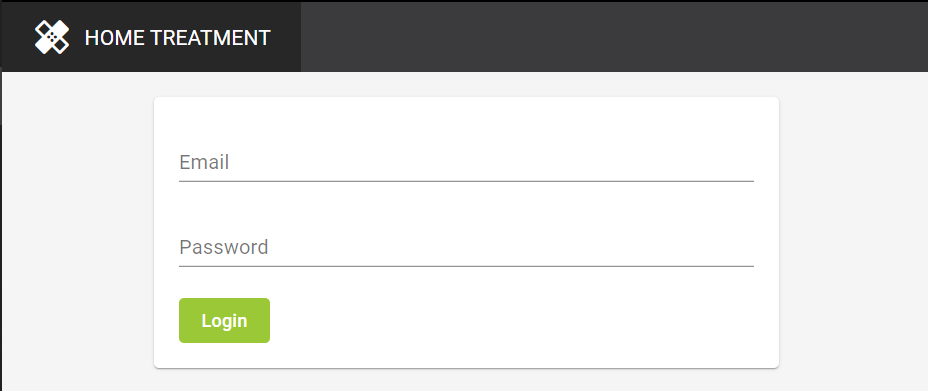
****

Рисунок 3.7 – Страница авторизации

После успешной авторизации пользователь автоматически будет направлен на основную страницу, для поиска заболеваний (рисунок 3.8). Для поиска нужного заболевания, необходимо заполнить фильтр с соответствующими симптомами. Для более точного поиска заболевания, можно вводить любое количество подходящих симптомов.

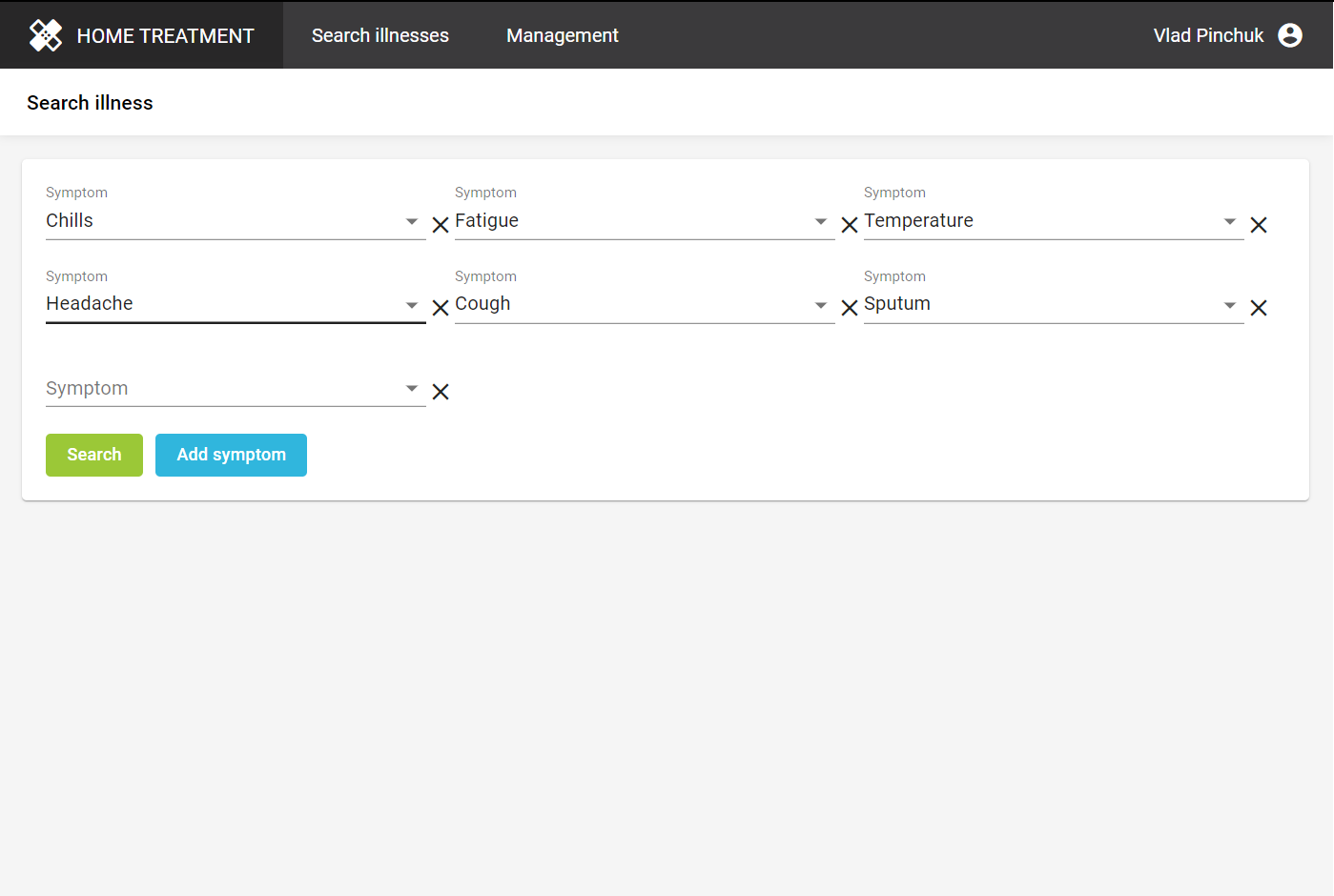
****

Рисунок 3.8 – Страница для поиска заболеваний

Для выбора симптомов, используется выпадающий список, со всеми доступными симптомами (рисунок 3.9). Для запуска поиска пользователь должен нажать кнопку «Search» или воспользоваться кнопкой «Enter» на клавиатуре.

После успешного поиска на странице появляется таблица со всеми заболеваниями подходящими, под заданный фильтр. Результирующая таблица представлена на рисунке 3.10. Таблица имеет две колонки: наименование заболевания и его рейтинг. Для просмотра подробной информации о заболевании нужно нажать на соответствующее наименование. После нажатия на наименование, в новой вкладе откроется страница с подробной информацией о заболевании.

A screenshot of a social media post

Description automatically generated

Рисунок 3.9 – Выпадающий список с симптомами

**A screenshot of a social media post

Description automatically generated**

Рисунок 3.10 – Результат поиска заболеваний

Для отображения более детальной информации о заболевании, создана отдельная страница (рисунок 3.11). На данной странице описана подробная информация о заболевании, а именно: подробное описание болезни, симптомы, рекомендации и список лекарственных средств, необходимых для лечения, данного заболевания (рисунок 3.12).

A screenshot of a social media post

Description automatically generated

Рисунок 3.11 – Полная информация о заболевании



Рисунок 3.12 – Список лекарственных средств

Для каждого заболевания пользователь может оставить отзыв о полезности и действительности представленных данных о заболевании. Для создания отзыва создана соответствующая страница (рисунок 3.13). На данной странице пользователь может оставить оценку для каждого раздела, а именно: подробное описание заболевания, симптомы, рекомендации и лекарственные средства. Также для полезности каждого отзыва, добавлено текстовое поле, где пользователь может подробно описать проблему с соответствующей информацией о заболевании.

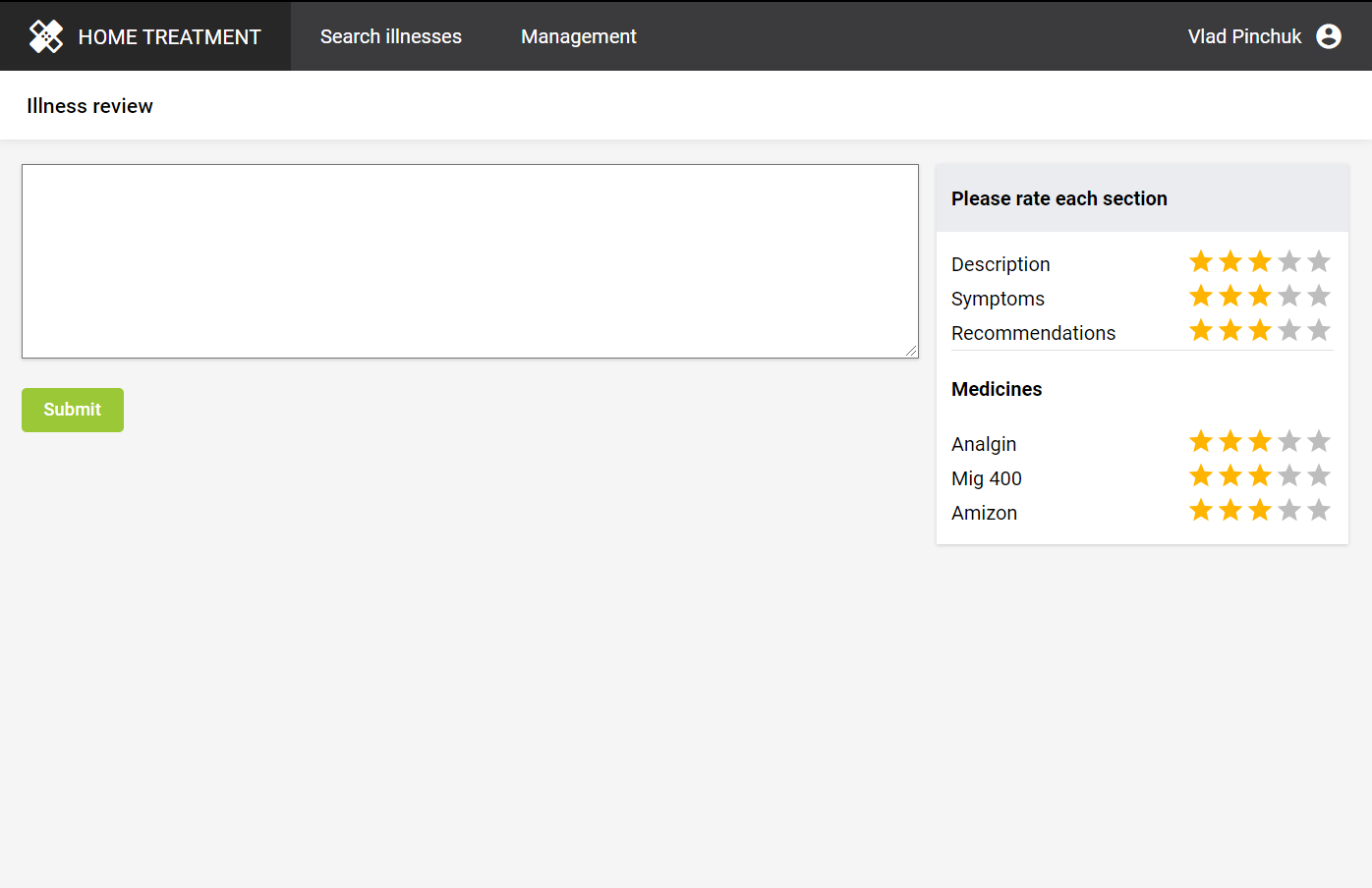


Рисунок 3.13 – Создание отзыва

Для создания нового заболевания, для экспертов создана отдельная страница. На рисунке 3.14 представлена страница для создания заболевания. Чтобы создать новое заболевание, нужно заполнить все обязательные поля. Для каждого заболевания можно добавить неограниченное количество рекомендации, лекарственных средств и симптомов. Для подробного описания заболевания, добавлен специальный редактор, который позволяет: редактировать текст, добавлять картинки, добавлять ссылки, добавлять видеозаписи и т.д.

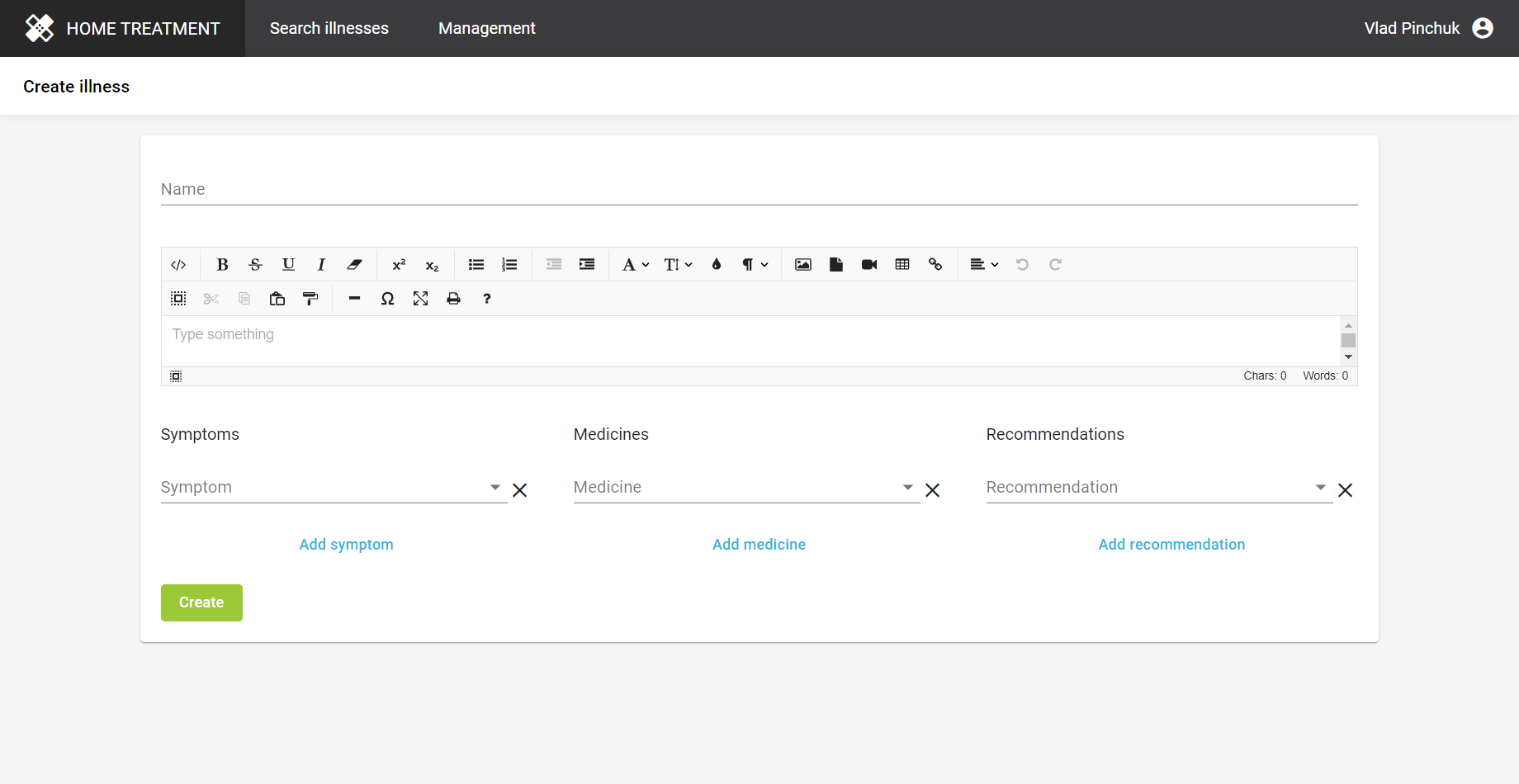


Рисунок 3.14 – Создание заболевания

Также, для управления всеми созданными заболеваниями, существует отдельная страница, которая позволяет выполнять основные операции. Данная страница представлена на рисунке 3.15. На данной странице в таблице отображаются все созданные заболевания. Таблица имеет следующие колонки: уникальный идентификатор, наименование, дата создания, дата последнего обновления и различные операции. Любое заболевание можно отредактировать, удалить или перейти к его анализу. Для быстрого перехода к созданию нового заболевания, была добавлена соответствующая кнопка.

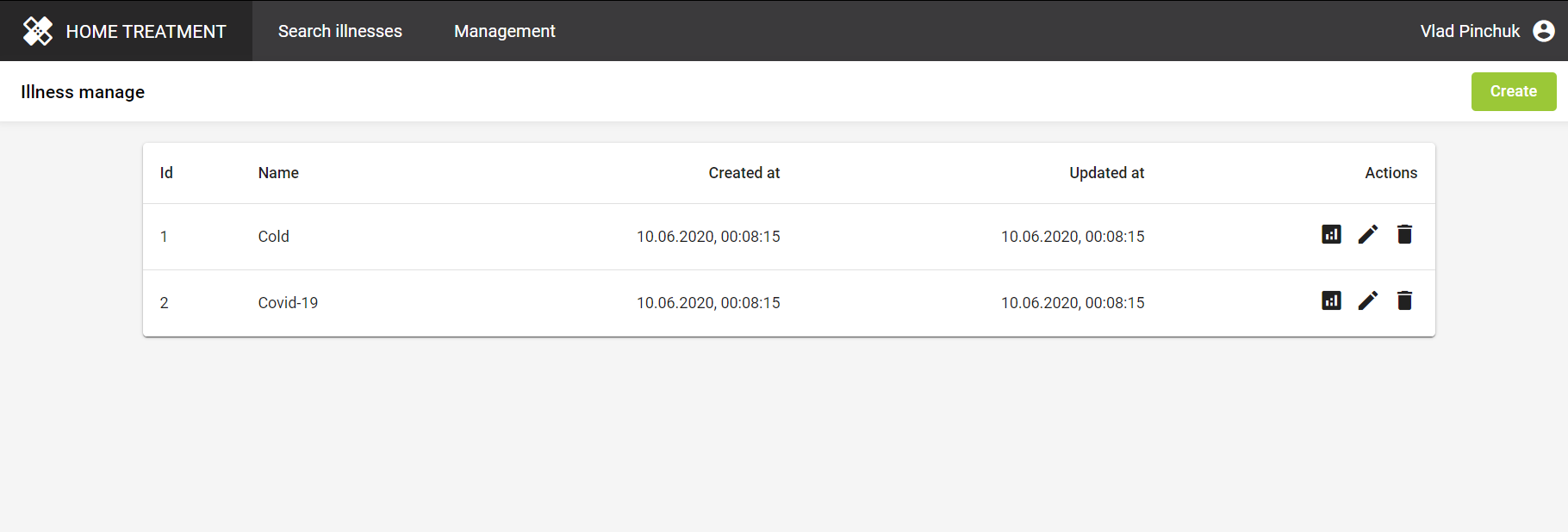


Рисунок 3.15 – Управление заболеваниями

Страница анализа состоит из двух блоков, а именно: графического отображения рейтинга заболевания и списка всех отзывов. На рисунке 3.16 представлен блок с отображение рейтинга в виде графиков. На первом графике отображается общий рейтинг заболевания за все время. На втором графике отображается рейтинг для конкретного лекарственного средства. Для выбора конкретного лекарственного средства, создан выпадающий список со всеми лекарственными средствами, добавленными к этому заболеванию. Также добавлен специальный инструмент для масштабирования графика.

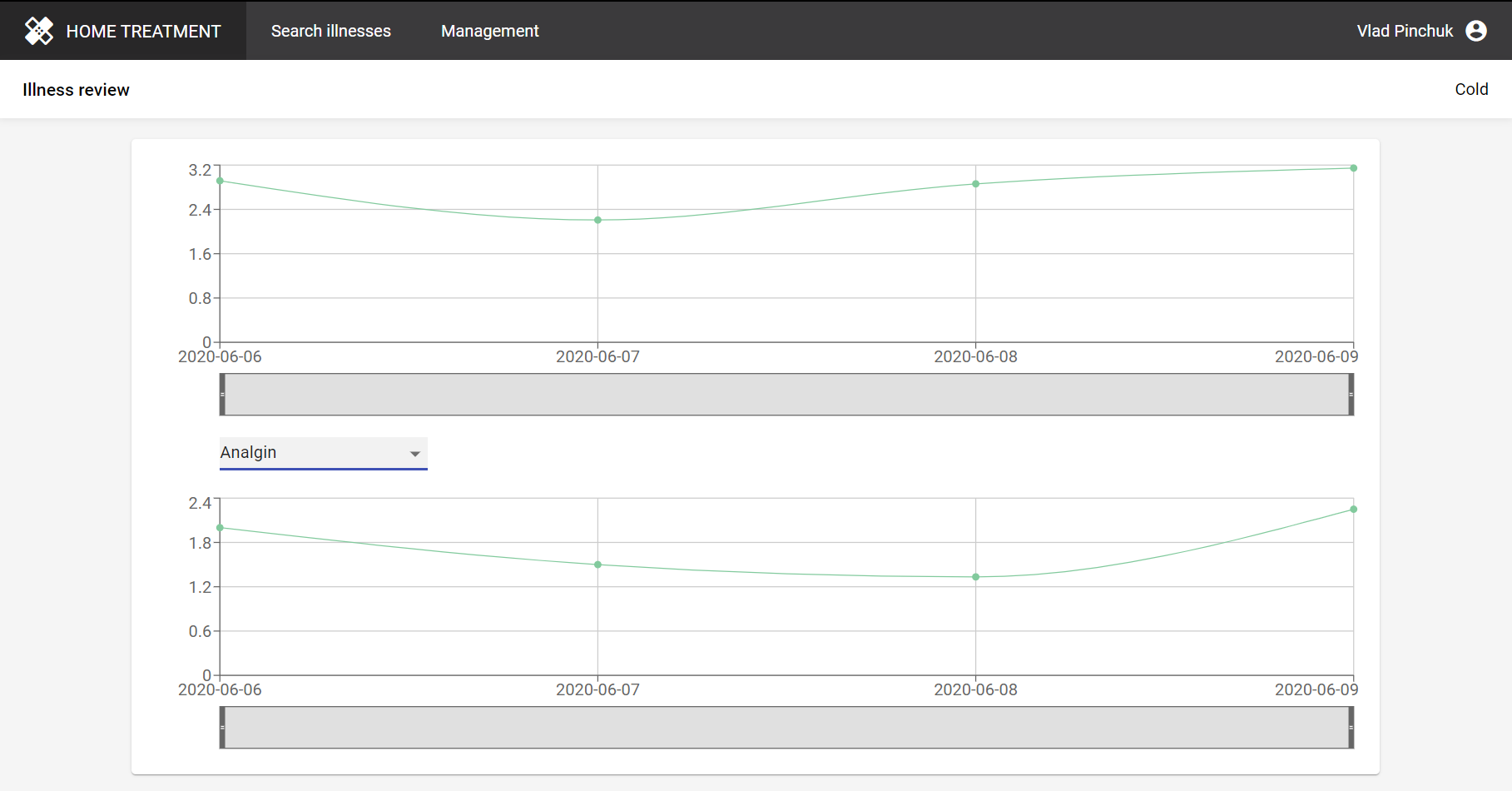


Рисунок 3.16 – Отображение рейтинга в виде графиков

На рисунке 3.17 представлен блок с отображением всех отзывов. Для отображения полной информации отзыва были применены расширяющиеся панели.

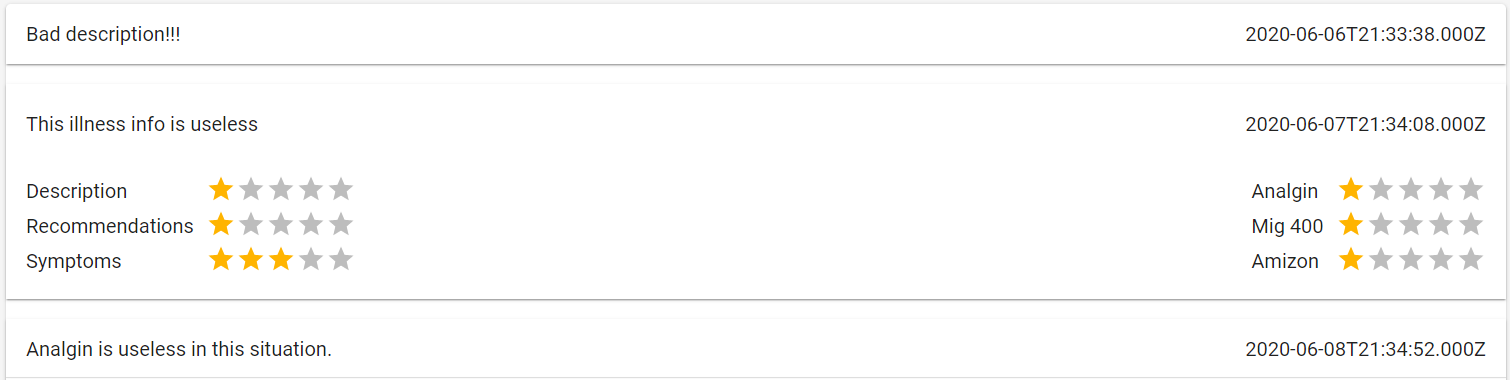


Рисунок 3.17 – Отображение всех отзывов

Приложение имеет простой, удобный и отзывчивый дизайн, понятный пользователю. Дизайн был разработан с учетом современных требований в данной области. Цвета всех элементов на странице подобраны в соответствии с опытом пользователей.

Такжеприложение имеет адаптивный дизайн, таким образом приложение поддерживает различные разрешения экрана и поэтому может быть запущено на различных устройствах, таких как: стационарный персональный компьютер, ноутбук, планшет и мобильные устройства с меньшим разрешением до 600 пикселей.

**4 ВЕРИФИКАЦИЯ И ОПЫТНАЯ ЭКСПЛУАТАЦИЯ**

**ПРОГРАММНОГО КОПМЛЕКСА**

Тестирование программного обеспечения- проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.

Для проверки соответствия программного продукта заявленным требованиям производится процесс верификации, или предоставления объективных свидетельств соответствия продукта поставленным требованиям.

В первую очередь приложение должно выполнять поставленные задачи, описанные в пункте 1.3.

Для каждой страницы приложения составлены определенные наборы тестов. Тесты составлены только для основного функционала, отказ которого будет критичным, блокирующим для всего приложения.

Перед тестированием приложения, база данных заполняется начальными данными, предназначенными для тестирования.

Самая первая страница, где оказывается пользователь, является страница авторизации, поэтому тестирование данной страницы является наиболее важной частью всего тестирования. Тест-кейсы для данной страницы представлены в таблице 4.1.

Таблица 4.1 – Тест-кейсы для страницы с авторизацией

|  |  |  |  |
| --- | --- | --- | --- |
| № | Краткое описание | Шаги для воспроизведения | Ожидаемый результат |
| 1 | 2 | 3 | 4 |
| 1 | Проверка успешной авторизации | 1. Перейти на страницу для авторизации  2. Ввести существующий адрес электронной почты и пароль  3. Нажать кнопку «Login» | Приложение автоматически должно перенаправить пользователя на страницу с поиском заболеваний. |

Продолжение таблицы 4.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 2 | Проверка на неверный формат адреса электронной почты | 1. Перейти на страницу для авторизации  2. Ввести любое значение не соответствующее формату адресу электронной почты  3. Потерять фокус с поля | Под полем ввода адреса электронной почты должна появиться ошибка «Enter valid email». |
| 3 | Проверка на отправку формы с пустыми полями | 1. Перейти на страницу для авторизации  2. Нажать кнопку «Login» | Под каждым полем должна появиться ошибка «Should not be empty». |
| 4 | Проверка на несуществующий адрес электронной почты или пароль | 1. Перейти на страницу для авторизации  2. Ввести несуществующий адрес электронной почты в нужном формате  3. Ввести несуществующий пароль  4. Нажать кнопку «Login» | Возле кнопки «Login» должна появиться ошибка «Wrong email or password». |

На рисунке 4.1 и 4.2 представлен вывод соответствующих ошибок. Как можно видеть каждое отрицательное действие сопровождается выводом соответствующей ошибки. Каждая ошибка появляется под соответствующим полем, что дает понять пользователю, где именно ошибка.

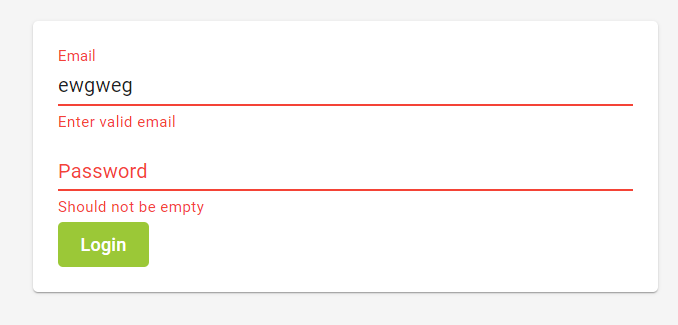
****

Рисунок 4.1 – Вывод ошибок под полями ввода

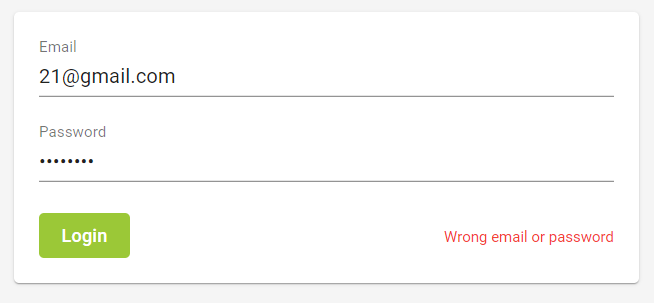
****

Рисунок 4.2 – Вывод ошибки о неверно введенных данных

Следующая страница на которую попадает пользователь, после успешной авторизации, является страница с поиском заболеваний. Тест-кейсы для данной страницы представлены в таблице 4.2.

Таблица 4.2 – Тест-кейсы для страницы с поиском заболеваний

|  |  |  |  |
| --- | --- | --- | --- |
| № | Краткое описание | Шаги для воспроизведения | Ожидаемый результат |
| 1 | 2 | 3 | 4 |
| 1 | Проверка поиска с одним симптомом | 1. Перейти на страницу с поиском заболеваний  2. Выбрать симптом «Temperature»  3. Нажать кнопку «Search» | На странице появляется таблица с двумя записями: Cold и Covid-19. |

Продолжение таблицы 4.2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 2 | Проверка поиска с двумя симптомами | 1. Перейти на страницу с поиском заболеваний  2. Выбрать симптом «Cough» и «Headache»  3. Нажать кнопку «Search» | На странице появляется таблица с одной записью: Flu. |
| 3 | Проверка поиска без симптомов | 1. Перейти на страницу с поиском заболеваний  2. Нажать кнопку «Search» | Возле кнопки «Search» должна появиться ошибка «Please select at least one symptom». Таблица не появляется. |
| 4 | Проверка на добавление четырех симптомов | 1. Перейти на страницу с поиском заболеваний  2. Нажать кнопку «Add symptom» три раза | В фильтре должно появиться четыре поля для выбора симптомов. Четвертое поле должно находиться на следующей строке. |

Фильтр выдает ошибку, если не выбрать ни один симптом. Данная проверка сделана с учетом на то, что записей заболеваний в базе данных будет огромное количество, и чтобы не доставать все записи разом, что может занять неограниченное количество времени, была добавлена данная проверка. Вывод ошибки представлен на рисунке 4.3.

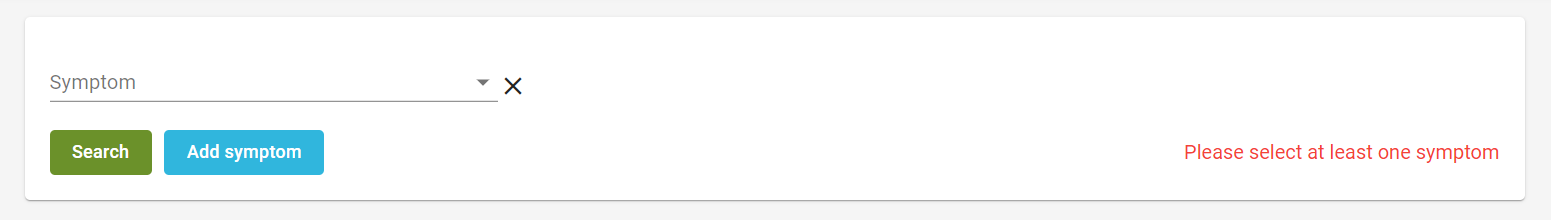


Рисунок 4.3 – Вывод ошибки в фильтре

Также в детальном тестировании нуждается страница создания заболевания, поскольку данная страница обладает достаточно сложной логикой. Тест-кейсы для данной страницы представлены в таблице 4.3.

Таблица 4.3 – Тест-кейсы для страницы с созданием заболевания

|  |  |  |  |
| --- | --- | --- | --- |
| № | Краткое описание | Шаги для воспроизведения | Ожидаемый результат |
| 1 | 2 | 3 | 4 |
| 1 | Проверка на создание заболевания с пустыми полями | 1. Перейти на страницу с созданием заболевания  2. Нажать кнопку «Create» | Под каждым полем должна появиться ошибка «Should not be empty». |
| 2 | Проверка на создание заболевания с уже существующим именем | 1. Перейти на страницу с созданием заболевания  2. В поле «Name» ввести «Cold» | Под полем «Name» должна появиться ошибка «Illness name should be uniq» |

Продолжение таблицы 4.3

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 3 | Проверка на создание заболевания без назначения лекарственных средств | 1. Перейти на страницу с созданием заболевания  2. Заполнить поля «Name», «Description», «Symptoms», «Recommendations» любыми подходящими значениями  3. Нажать на крестик в первом поле с лекарственным средством  4. Нажать кнопку «Create» | В колонке с «Medicines» появится ошибка «Please select at least one medicine». |
| 4 | Проверка на успешное создание заболевания | 1. Перейти на страницу с созданием заболевания  2. Заполнить все поля с любыми подходящими значениями  3. Нажать кнопку «Create» | Приложение должно автоматически перенести пользователя на страницу для управления заболеваниями. На данной странице, в таблице, должно появиться созданное ранее заболевание. При повторном переходе на страницу с созданием заболевания, форма для создания должна быть в начальном состоянии. |

Все поля в данной форме являются обязательными к заполнению. Вывод ошибок с пустыми полями представлен на рисунке 4.4. Также дополнительно, имя заболевания должно быть уникальным. Если будет введено уже существующее имя, то появится соответствующая ошибка (рисунок 4.5).

Поля с множественным добавлением имеют проверку, на выбор хотя бы одного значения. На рисунке 4.6 приведен пример с полем «Medicines». Аналогичная проверка присутствует и для полей «Symptoms» и «Recommendations».

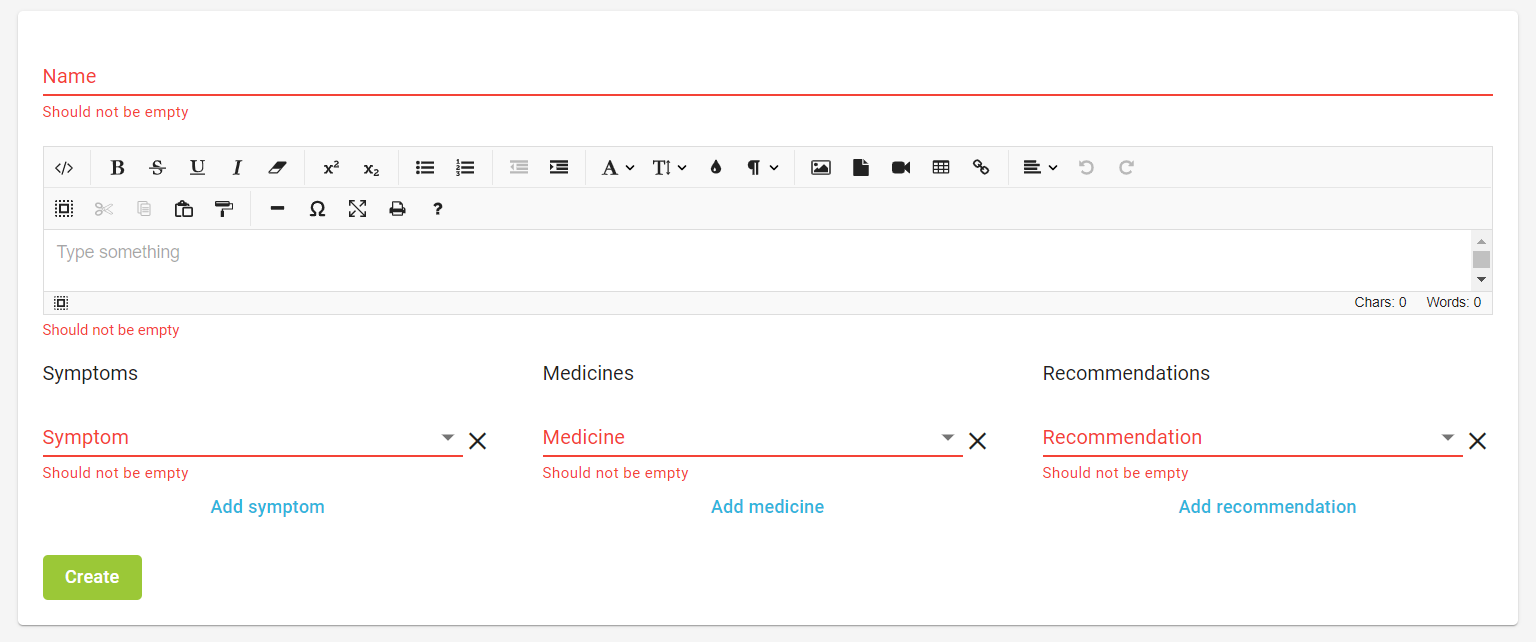
****

Рисунок 4.4 – Вывод ошибок в форме с пустыми полями

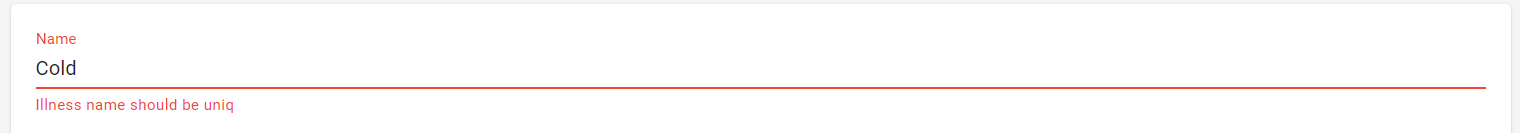
****

Рисунок 4.5 – Вывод ошибки об уникальности имени

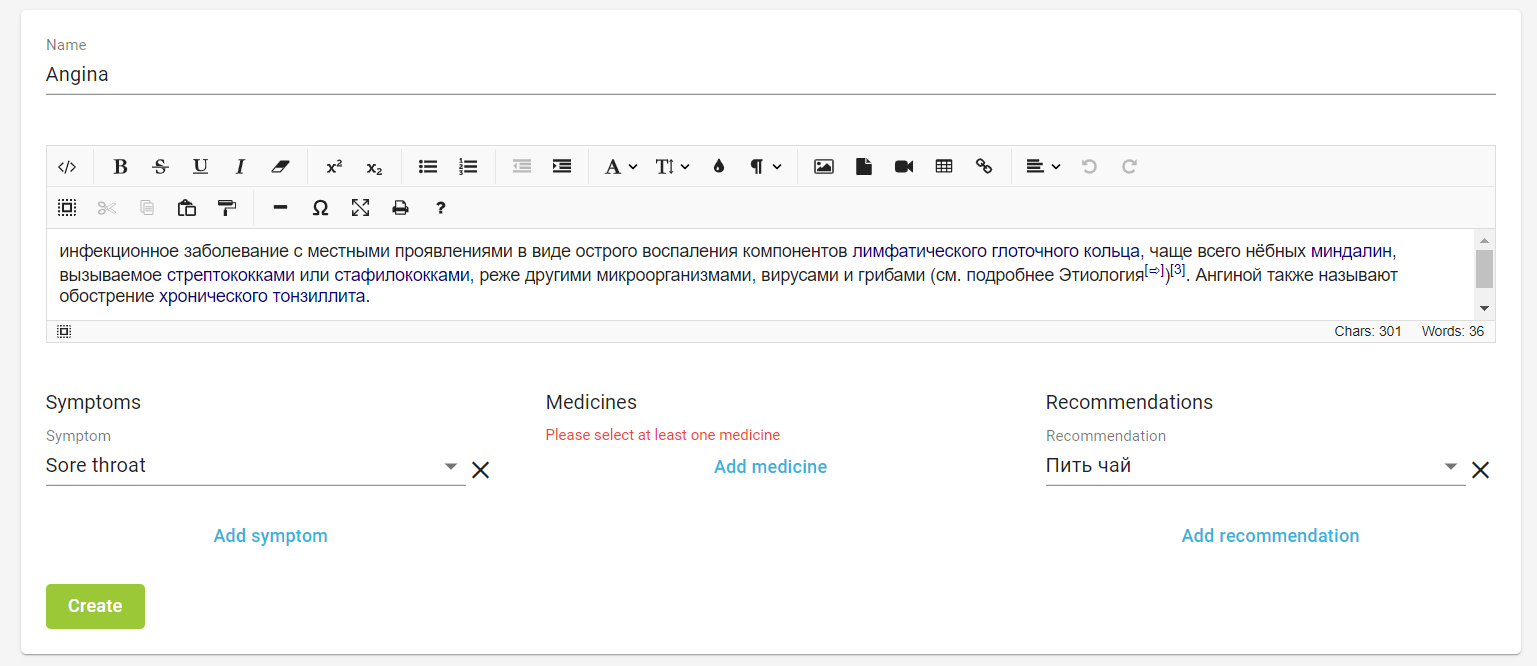
****

Рисунок 4.6 – Вывод ошибки о не выбранном лекарственном средстве

**5 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ДИПЛОМНОЙ РАБОТЫ**

[**5.1 Расчет общей трудоемкости разработки программного обеспечения**](#_Toc516728097)

Общий объем программного обеспечения (ПО) (Vo) определяется исходя из количества и объема функций, реализуемых программой, по каталогу функций ПО в соответствии с таблицей 1.1 [5] по формуле (5.1):

, (5.1)

где Vi – объем отдельной функции ПО;

n - общее число функций.

Анализируя разработанную программу, уточненный объем ПО (Vy) определяем по формуле 5.2:

, (5.2)

где Vyi – уточненный объем отдельной функции ПО в строках исходного кода (LOC).

В ходе уточнения объема функций проведены следующие операции:

1. уменьшен объем функций:
   * контроль, предварительная обработка и ввод информации;
   * обработка входного языка и формирование таблиц;
   * генерация структуры базы данных;
   * формирование базы данных;
   * манипулирование данными;
   * организация поиска в базе данных;
   * обработка ошибочных сбойных ситуаций;
   * расчетные задачи;
   * расчет показателей.
2. увеличен объем функций:

* организация ввода информации;
* обработка наборов и записей базы данных;
* графический вывод результатов*.*

Сравнение исходного и уточненных объемов строк исходного кода представлены в таблице 5.1.

Таблица 5.1 – Каталог функций программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| Код функций | Наименование (содержание) функций | Объем функции строк исходного кода (LOC) | |
| По каталогу (V0) | уточненный (Vy) |
| 1 | 2 | 3 | 4 |
| Ввод, анализ входной информации | | | |
| 101 | Организация ввода информации | 140 | 200 |
| 102 | Контроль, предварительная обработка и ввод информации | 370 | 210 |
| 104 | Обработка входного языка и формирование таблиц | 760 | 350 |
| Формирование, ведение и обслуживание базы данных | | | |
| 201 | Генерация структуры базы данных | 2300 | 780 |
| 202 | Формирование базы данных | 1240 | 410 |
| 203 | Обработка наборов и записей базы данных | 1070 |  |
| 206 | Манипулирование данными | 3100 | 650 |
| 207 | Организация поиска в базе данных | 2830 |  |
| Управление ПО, компонентами ПО и внешними устройствами | | | |
| 506 | Обработка ошибочных сбойных ситуаций | 940 | 180 |
| Расчетные задачи, формирование и вывод на внешние носители  документов сложной формы и файлов | | | |

Продолжение таблицы 5.1

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 702 | Расчетные задачи | 3200 | 800 |
| 703 | Расчет показателей | 430 | 150 |
| 707 | Графический вывод результатов | 800 | 920 |
|  | Итого: |  |  |

Согласно таблице 1.2 [5], разработанное в ходе выполнения дипломной работы программное обеспечение по своим характеристикам относится к моделированию объектов и процессов, категория 2.

На основании принятого к расчету (уточненного) объема (Vy) и категории сложности ПО определяется нормативная трудоемкость ПО (Тн) 1.3 [5] выполняемых работ, представлены в таблице 5.2.

Таблица 5.2 – Нормативная трудоемкость на разработку ПО (Тн)

|  |  |  |
| --- | --- | --- |
| Уточненный объем, Vу | 2-я категория сложности ПО | Номер нормы |
|  | 299 | 52 |

Дополнительные затраты труда, связанные с повышением сложности разрабатываемого ПО, учитываются посредством коэффициента повышения сложности ПО (Кс).

Кс рассчитывается по формуле:

, (5.3)

где Кi – коэффициент соответствующий степени повышения сложности [5];

n – количество учитываемых характеристик.

Новизна разработанного ПО определяется путем экспертной оценки данных, полученных при сравнении характеристик разрабатываемого ПО с имеющимися аналогами. Влияние фактора новизны на трудоемкость учитывается путем умножения нормативной трудоемкости на соответствующий коэффициент, учитывающий новизну ПО (Кн). В соответствии с таблицей 2.2 [5] разработанная программа обладает категорией новизны В, а значение Кн = 0,63.

Современные технологии разработки компьютерных программ предусматривают широкое использование коробочных продуктов (пакетов, модулей, объектов). Степень использования в разрабатываемом ПО стандартных модулей определяется их удельным весом в общем объеме ПО.

В данном программном комплексе используется до 60% стандартных модулей, что соответствует значению коэффициента КТ = 0,65 таблица 2.3 [5].

Программный комплекс разработан на языке TypeScript, что согласно таблице 2.4 [5] соответствует коэффициенту, учитывающему средства разработки ПО, Кур = 0,6.

Значение коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО, определяются с учетом установленной категории новизны ПО согласно таблице 2.5 [5].

При этом сумма значений коэффициентов удельных весов всех стадий в общей трудоемкости равна единице. Значения коэффициентов приведены в таблице 5.3.

Таблица 5.3 – Значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО

|  |  |  |  |
| --- | --- | --- | --- |
| Категория новизны ПО | С применением CASE-технологии | | |
| Стадии разработки ПО | | |
| ТЗ+ЭП+ТП | РП | ВН |
| Значения коэффициентов | | |
| КТЗ+ КЭП + КТП | КРП | КВН |
| B | 0,50 | 0,35 | 0,15 |

Нормативная трудоемкость ПО (Тн) выполняемых работ по стадиям разработки корректируется с учетом коэффициентов: повышения сложности ПО (Кс), учитывающих новизну ПО (Кн), учитывающих степень использования стандартных модулей (Кт), средства разработки ПО (Кур) и определяются по формулам:

для стадии ТЗ по формуле (5.4)

, (5.4)

для стадии ЭП формуле (5.5)

, (5.5)

для стадии ТП формуле (5.6)

, (5.6)

для стадии РП формуле (5.7)

, (5.7)

для стадии ВН формуле (5.8)

, (5.8)

где и – значения коэффициентов удельных весов трудоемкости стадий разработки ПО в общей трудоемкости ПО.

Коэффициенты вводятся на всех стадиях разработки, а коэффициент вводится только на стадии РП.

Общая трудоемкость разработки ПО (ТO) определяется суммированием нормативной (скорректированной) трудоемкости ПО по стадиям разработки по формуле (5.9)

, (5.9)

где Тyi – нормативная (скорректированная) трудоемкость разработки ПО на i-й стадии (чел.-дн.);

n – количество стадий разработки.

Результаты расчетов по определению нормативной и скорректированной трудоемкости ПО по стадиям разработки и общую трудоемкость разработки ПО (ТO) представлены в таблице 5.4.

Таблица 5.4 – Расчет общей трудоемкости разработки ПО

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Показатели | Стадии разработки | | | | | Итого |
| ТЗ | ЭП | ТП | РП | ВН |  |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Общий объем ПО (Vo), кол-во строк LOC | - | - | - | - | - |  |
| Общий уточненный объем ПО (Vy), кол-во строк LOC | - | - | - | - | - |  |
| Категория сложности разрабатываемого ПО | - | - | - | - | - | 2 |
| Нормативная трудоемкость разработки ПО (Тн), чел.-дн. | - | - | - | - | - | 310 |
| Коэффициент повышения сложности ПО (Кс) | 1,25 | 1,25 | 1,25 | 1,25 | 1,25 | - |
| Коэффициент, учитывающий новизну ПО (Кн) | 0,63 | 0,63 | 0,63 | 0,63 | 0,63 | - |
| Коэффициент, учитывающий степень использования стандартных модулей (Кт) | - | - | - | 0,65 | - | 0,65 |
| Коэффициент, учитывающий средства разработки ПО(Кур) | 0,6 | 0,6 | 0,6 | 0,6 | - | - |
| Коэффициенты удельных весов трудоемкости стадий разработки ПО (КТЗ, КЭП, КТП, КРП, КВН) | 0,50 | | | 0,35 | 0,15 | 1,0 |
| Распределение скорректированной (с учетом Кс, Кн, Кт, Кур) трудоемкости ПО по стадиям, чел.-дн. | 70 | | | 32 | 21 | - |
| Общая трудоемкость разработки ПО (То), чел.-дн. | - | - | - | - | - | 123 |

[**5.2 Расчет затрат на разработку программного продукта**](#_Toc516728098)

В состав затрат на разработку программного продукта входят следующие статьи расходов:

* затраты труда на создание программного продукта (затраты по основной, дополнительной заработной плате и соответствующие отчисления) (Зтр);
* затраты на изготовление эталонного экземпляра (Зэт);
* затраты на технологию (затраты на приобретение и освоение программных средств, используемых при разработке программного продукта; затраты на ПО, используемое как эталон) (Зтех);
* затраты на машинное время (расходы на содержание и эксплуатацию технических средств разработки, эксплуатации и сопровождения) (Змв);
* затраты на материалы (информационные носители) (Змат);
* затраты на энергию, на использование каналов связи (для отдельных видов);
* общепроизводственные расходы (затраты на управленческий персонал, на содержание помещений) (Зобщ\_пр);
* непроизводственные (коммерческие) расходы (затраты связанные с рекламой, поиском заказчиков, поставками конкретных экземпляров) (Знепр).

В таблице 5.5 приведены значения основных параметров, необходимых для расчета затрат на разработку программного продукта. Все данные, где не указан источник берутся из [5].

Таблица 5.5 – Параметры для расчета производственных затрат на разработку ПО

|  |  |  |
| --- | --- | --- |
| Параметр | Единица измерения | Значение |
| 1 | 2 | 3 |
| Базовая ставка | руб. | 185 |
| Разряд разработчика | – | 8 |
| Тарифный коэффициент | – | 1,57 |
| Коэффициент Кув | – | 1,6 |
| Норматив отчислений на доп. зарплату разработчиков (Ндоп) | % | 10 |
| Численность обслуживающего персонала | чел. | 1 |
| Разряд обслуживающего персонала | – | 2 |
| Тарифный коэффициент | – | 1,07 |

Продолжение таблицы 5.5

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| Средняя годовая ставка арендных платежей (CАР) (вычеслено по алгоритму на сайте *http://www.samsebeyurist.by/spravochnaya-informatsiya/stavki-i-velichiny/bazovaja-arendnaja-velichina*) | руб./м2 | 86,5 |
| Площадь помещения (S) | м2 | 10 |
| Количество ПЭВМ (QЭВМ) | шт. | 1 |
| Затраты на приобретение единицы ПЭВМ | руб. | 1000 |
| Стоимость одного кВт-часа электроэнергии (СЭЛ) (*http://www.gomelenergo.by/*) | руб. | 0,2185 |
| Коэффициент потерь рабочего времени (Кпот) | - | 0,2 |
| Затраты на технологию (Зтех) | руб. | – |
| Норматив общепроизводственных затрат (Ндоп) | % | 5 |
| Норматив непроизводственных затрат (Ннепр) | % | 5 |

Суммарные затраты на разработку ПО (Зр) определяются по формуле (5.10):

, (5.10)

Расходы на оплату труда разработчиков с отчислениями равны:

, (5.11)

где ЗПосн – основная заработная плата разработчиков, руб.;

ЗПдоп – дополнительная заработная плата разработчиков, руб.;

ТЧзп – сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.

Основная заработная плата разработчиков рассчитывается по формуле (4.12):

, (5.12)

где Сср\_час – средняя часовая тарифная ставка;

То – общая трудоемкость разработки, чел-час;

Кув – коэффициент, учитывающий доплаты стимулирующего характера.

Средняя часовая тарифная ставка определяется по формуле (5.13):

, (5.13)

где Счi - часовая тарифная ставка разработчика i – й категории;

ni – количество разработчиков i-й категории.

(5.14)

где Tcт – базовая ставка;

k – тарифный коэффициент.

.

.

Дополнительная заработная плата равна:

, (5.15)

где Ндоп – норматив отчислений на дополнительную заработную плату разработчиков.

Отчисления от основной и дополнительной заработной платы (отчисления на социальные нужды и обязательное страхование) рассчитываются по формуле (5.16):

, (5.16)

где Hзп – процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы (Hзп = 35 %).

Затраты машинного времени определяются по формуле:

, (5.17)

где Сч – стоимость 1 часа машинного времени (руб./ч.);

Кт – коэффициент мультипрограммности, показывающий распределение времени работы ЭВМ в зависимости от количества пользователей ЭВМ; Кт =1;

tэвм – машинное время ЭВМ, необходимое для разработки и отладки проекта (ч.).

Стоимость машино-часа определяется по формуле (5.18):

, (5.18)

где – затраты на заработную плату обслуживающего персонала с учетом всех отчислений, (руб. в год);

– стоимость аренды помещения под размещение вычислительной техники, (руб. в год);

– амортизационные отчисления за год, (руб. в год);

– затраты на электроэнергию, (руб. в год);

– затраты на материалы, необходимые для обеспечения нормальной работы ПЭВМ (вспомогательные), (руб. в год);

– затраты на текущий и профилактический ремонт ЭВМ (руб. в год);

– прочие затраты, связанные с эксплуатацией ПЭВМ. (руб. в год);

– действительный фонд времени работы ЭВМ, (час/год).

Все статьи затрат формируются в расчете на единицу ПЭВМ.

Затраты на заработную плату обслуживающего персонала (ЗПобсл) определяются по формуле (5.19):

, (5.19)

, (5.20)

, (5.21)

, (5.22)

где – основная заработная плата обслуживающего персонала, руб.;

– дополнительная заработная плата обслуживающего персонала, руб.;

– сумма отчислений от заработной платы (социальные нужды, страхование от несчастных случаев), руб.;

– количество обслуживаемых ПЭВМ, шт.;

– базовая ставка, руб.;

– численность обслуживающего персонала, чел.;

– процент дополнительной заработной платы обслуживающего персонала от основной;

– процент отчислений на социальные нужды и обязательное страхование от суммы основной и дополнительной заработной платы.

Годовые затраты на аренду помещения (ЗАР) определяются по формуле (5.23):

, (5.23)

где CАР – средняя годовая ставка арендных платежей, руб./м2;

S – площадь помещения, м2.

Сумма годовых амортизационных отчислений (ЗАМ) определяется по формуле (5.24):

, (5.24)

где – затраты на приобретение (стоимость) единицы ПЭВМ, руб;

– коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, = 12-13 % от ;

(1+ ) – балансовая стоимость ЭВМ, руб;

– норма амортизации, %.

Стоимость электроэнергии, потребляемой за год, (ЗЭП) определяется по формуле (5.25):

, (5.25)

где М – паспортная мощность ПЭВМ, (кВт), М = 0,5 кВт;

– стоимость одного кВт-часа электроэнергии, руб;

А – коэффициент интенсивного использования мощности, А=0,98…0,9.

Действительный годовой фонд времени работы ПЭВМ (FЭВМ) рассчитывается:

, (5.26)

где Дг – общее количество дней в году, Дг = 365 дней;

Двых, Дпр – число выходных и праздничных дней в году, Двых + Дпр = 111 дн.;

Fсм – продолжительность 1 смены, Fсм = 8 часов;

Ксм – коэффициент сменности, т.е. количество рабочих смен ЭВМ, Ксм = 1;

Кпот – коэффициент, учитывающий потери рабочего времени, связанные с профилактикой и ремонтом ЭВМ, Кпот = 0,15- 0,30.

Затраты на материалы (ЗВМ), необходимые для обеспечения нормальной работы ПЭВМ составляют около 1% от балансовой стоимости ЭВМ и определяются:

, (5.27)

где Зприобр – затраты на приобретение (стоимость) ЭВМ, (руб);

Кдоп – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, Кдоп = 12-13 % от Зприобр;

Кмз – коэффициент, характеризующий затраты на вспомогательные материалы (Кмз = 0,01).

Затраты на текущий и профилактический ремонт (ЗТР) принимаются равными 5% от балансовой стоимости ЭВМ:

, (5.28)

где Ктр – коэффициент, характеризующий затраты на текущий и профилактический ремонт (Кмз = 0,05).

Прочие затраты, связанные с эксплуатацией ЭВМ (ЗПР) состоят из амортизационных отчислений на здания, стоимости услуг сторонних организаций и составляют 5 % от балансовой стоимости:

, (5.29)

где Кпр – коэффициент, характеризующий размет прочих затрат, связанных с эксплуатацией ЭВМ (Кпр = 0,05).

Для расчета машинного времени ЭВМ (tэвм в часах), необходимого для разработки и отладки проекта, следует использовать формулу (5.30):

, (5.30)

где tРП – срок реализации стадии «Рабочий проект» (РП), 20 дней;

tВН – срок реализации стадии «Ввод в действие» (ВП), 8 дней;

Fсм – продолжительность рабочей смены, (ч.), Fсм = 8 ч.;

Ксм – количество рабочих смен, Ксм = 1.

Расчет затрат на изготовление эталонного экземпляра (Зэт) осуществляется по формуле (5.31):

, (5.31)

где Кэт – коэффициент, учитывающий размер затрат на изготовление эталонного экземпляра, (Кэт  = 0,05).

При написании дипломной работы было использовано только бесплатное ПО, поэтому затраты на технологию (Зтех) будут нулевыми.

Затраты на материалы (носители информации и пр.), необходимые для обеспечения нормальной работы ПЭВМ рассчитываются следующим образом:

, (5.32)

где Зприобр – затраты на приобретение ЭВМ, руб;

Кдоп – коэффициент, характеризующий дополнительные затраты, связанные с доставкой, монтажом и наладкой оборудования, Кдоп = 12-13 % от Зприобр;

Кмз – коэффициент, характеризующий затраты материалы (Кмз = 0,01).

Общепроизводственные затраты рассчитываются по формуле (5.33):

, (5.33)

где Ндоп – норматив общепроизводственных затрат.

Принимаем

Непроизводственные затраты рассчитываются по формуле (5.34):

, (5.34)

где Ннепр – норматив непроизводственных затрат.

Принимаем

Итого получаем суммарные затраты на разработку:

Результаты расчетов приведены в таблице 5.6.

Таблица 5.6 – Расчет суммарных затрат на разработку ПО, руб.

|  |  |
| --- | --- |
| Статья затрат | Итого |
| 1 | 2 |
| Затраты на оплату труда разработчиков (Зтр), руб. |  |
| Основная заработная плата разработчиков, руб. |  |
| Дополнительная заработная плата разработчиков, руб. |  |
| Отчисления от основной и дополнительной заработной платы, руб. |  |
| Затраты машинного времени (Змв), руб. |  |
| Стоимость машино-часа, руб/ч |  |
| Сумма годовых амортизационных отчислений, руб. |  |

Продолжение таблицы 5.6

|  |  |
| --- | --- |
| 1 | 2 |
| Действительный годовой фонд времени работы ПЭВМ, ч. |  |
| Затраты на текущий и профилактический ремонт, руб. |  |
| Прочие затраты, связанные с эксплуатацией ЭВМ, руб. |  |
| Машинное время ЭВМ, ч. |  |
| Затраты на изготовление эталонного экземпляра (Зэт), руб. |  |
| Затраты на технологию | 0 |
| Затраты на материалы (Змат), руб. |  |
| Суммарные затраты на разработку ПО () |  |

[**5.3 Формирование цены при создании программного обеспечения**](#_Toc516728099)

Оптовая цена ПП () определяется следующим образом:

, (5.35)

, (5.36)

где – себестоимость ПО, руб.; – прибыль от реализации ПП, руб.; – уровень рентабельности ПП, % (Ур = 30 %).

Прогнозируемая отпускная цена ПП без НДС рассчитывается:

, (5.37)

Налог на добавленную стоимость () рассчитывается по формуле

, (5.38)

где – ставка налога на добавленную стоимость, %, = 20 %

Розничную цену на ПП () можно определить следующим образом:

, (5.39)

где – торговая наценка при реализации программного обеспечения через специализированные магазины (торговых посредников), ее значение принимается равной 10%.

Таблица 5.7 – Расчет формирования цены на разработку программы

|  |  |
| --- | --- |
| Наименование статьи расходов | Значение |
| 1 | 2 |
| Полная себестоимость (суммарные затраты на разработку ПО) |  |
| Прибыль от реализации ПО |  |
| Оптовая цена ПП |  |
| Налог на добавленную стоимость |  |
| Отпускная цена ПО с НДС |  |
| Розничная цена |  |

[**5.4 Расчет эффекта от внедрения программного обеспечения**](#_Toc516728100)

Расчет эффекта (прибыли) от реализации (внедрения) разработанного в ходе выполнения дипломной работы программного комплекса может вестись двумя путями:

* на основе сравнения с уже существующим программным обеспечением;
* на основе экспертных оценок эффективности работы ПО в реальных условиях эксплуатации.

Методика первого вида расчета зависит от:

* коммерческого или некоммерческого характера разработки;
* области применения разработки, прежде всего в производственной или непроизводственной сфере;
* стадии внедрения и степени ее реализации (опытная отладка или эксплуатация, полный цикл внедрения и т.д.);
* наличия прямых эффектов (для коммерческих проектов) или (и) косвенных (для некоммерческих проектов);
* наличие эффекта в виде роста количества или (и) качества выпускаемой продукции, выполняемых работ или услуг; снижение затрат на производство или маркетинг; снижение затрат на компьютерно-информационные работы или услуги и т.д.

Эффект (прибыль) может просчитываться по формуле:

, (5.40)

где Збаз – текущие и инвестиционные затраты по базовому варианту, включающие затраты на приобретение продукта (цену), его эксплуатацию;

З – текущие и инвестиционные затраты по варианту, предложенному студентом-дипломником,

По результатам изучения рыночных цен, программных продуктов схожего функционального назначения нами было установлен аналог стоимостью 9700$, или 23377 рублей.

.

На основе рассчитанного эффекта от разработки ПО следует нами рассчитаны следующие итоговые показатели, характеризующие экономическую эффективность проекта:

Рентабельность затрат (З) или инвестиций (И) на новую информационную технологию, программных продукт:

, (5.41)

Срок окупаемости затрат (инвестиций):

, лет, (5.42)

Т.к. срок окупаемости составляет меньше одного календарного года, то проведение динамической оценки (расчет динамических показателей эффективности) не целесообразно.

Годовой экономический эффект определяется:

, (5.43)

где Рбаз – рентабельность затрат (инвестиций) базового варианта, руб 25%.

Все данные приведены в итоговой таблице 5.8

Таблица 5.8 – Технико-экономические показатели проекта

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование показателя | Единица измерения | Проектный вариант |
| 1 | 2 | 3 | 4 |
| Показатели затрат на разработку | | | |
| 2 | Общая трудоемкость разработки ПО | чел.- дн |  |
| 3 | Затраты на разработку программы | руб. |  |
| 3.1 | Затраты на оплату труда разработчиков | руб. |  |
| 3.2 | Затраты машинного времени | руб. |  |
| 3.3 | Затраты на изготовление эталонного экземпляра | руб. |  |
| 3.4 | Затраты на технологию | руб. | 0 |
| 3.5 | Затраты на материалы | руб. |  |
| 3.6 | Общепроизводственные затраты | руб. |  |
| 3.7 | Непроизводственные (коммерческие) затраты | руб. |  |
| Показатели стоимости | | | |
| 5 | Отпускная цена ПП с НДС | руб. |  |

Продолжение таблицы 5.8

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 6 | Розничная цена ПП | руб. |  |
| Показатели экономической эффективности | | | |
| 7 | Рентабельность затрат | % |  |
| 8 | Простой срок окупаемости проекта | лет | 0,49 |
| 9 | Годовой экономический эффект | руб. |  |

Таким образом, по результатам проведенной мною оценки установлено, что реализация проекта обоснована и является экономически целесообразной. Об этом свидетельствуют следующие показатели: срок окупаемости меньше года при размере годового экономического эффекта рублей с уравнением рентабельности Р = 203,3%.

**6 ОХРАНА ТРУДА И ТЕХНИКА БЕЗОПАСНОСТИ. ЗАЩИТА ОТ ЭЛЕКТРОМАГНИТНЫХ ИЗЛУЧЕНИЙ**

Электромагнитное излучение есть практически повсеместно. Многие считают, что электромагнитное излучение есть только лишь в электроустановках. Но это далеко не так. Электромагнитное излучение преследует нас везде: дома, на работе, на улице. Источниками электромагнитного излучения, помимо электрических сетей, является практически вся бытовая техника, в том числе различные электронные устройства: радиоаппаратура, мобильные телефоны, гаджеты и множество других электрических приборов [6].

Даже на улицах города, где, казалось бы, нет электромагнитного излучения, источниками такового является электрифицированный транспорт, силовые сети, сети уличного освещения и др. Рассмотрим, какое влияние оказывают те или иные источники электромагнитного излучения на организм человека.

Для начала отметим такой параметр, как предельно допустимая доза электромагнитного излучения для человека – она составляет 0,2 мкТл [7]. Теперь отметим среднее значение электромагнитных излучений различных электрических приборов и устройств, с которыми человек сталкивается в повседневной жизни.

Компьютер – неотъемлемый элемент в доме каждой семьи. В девяти домах из десяти есть компьютер или другая компьютерная техника (ноутбук, планшет и др.). Данное устройство является источником [электромагнитного излучения](http://electricalschool.info/spravochnik/electroteh/1874-jelektromagnitnye-volny.html) величиной до 100 мкТл. Несложно посчитать, что человек, находясь в непосредственной близости к компьютеру, подвергается электромагнитному излучению, которое в 500 раз превышает допустимое значение.

Практически такой же уровень электромагнитного излучения генерируется микроволновой печью. Даже обычная настольная лампа является источником электромагнитного излучения, которое в 4-5 раз превышает допустимое значение. В данном случае источником излучения является провод, питающий лампу.

Также следует отметить вредное воздействие мобильных телефонов и других гаджетов и электронных устройств. Электромагнитное излучение от данных аппаратов достигает 50 мкТл, что в 250 раз превышает допустимое значение.

Электрифицированный транспорт является одним из наиболее сильных источников электромагнитного излучения. Поездка в трамвае или троллейбусе сопровождается воздействием на организм человека электромагнитного излучения значением 150-200 мкТл. При чем, в метро значение электромагнитного излучения на порядок выше и оно составляет 300 мкТл.

Даже на отдыхе, где, казалось бы, человек находится вдали от источников электромагнитного излучения, но он также подвергается электромагнитному излучению. Источником электромагнитного излучения в данном случае являются высоковольтные линии электропередач, которые пересекают окружающую местность вдоль и в поперек.

Все приборы и устройства, получающие питание от электрической сети, в той или иной мере являются источниками электромагнитного излучения. Получается, что человек, проживающий в современных условиях, практически всегда подвергается электромагнитному излучению. Поэтому вопрос защиты организма от воздействия электромагнитного излучения в наше время является особенно актуальным. Рассмотрим основные меры снижения негативного воздействия электромагнитного излучения на организм человека [9].

Один из наиболее эффективных способов защиты от негативного воздействия электромагнитного излучения является применение специальных приборов, которые позволяют нейтрализовать данное излучение и максимально минимизировать его негативное воздействие на организм человека [8].

Максимальное сокращение времени пребывания в зоне действия электромагнитного излучения является одним из наиболее эффективных способов защиты организма от негативного воздействия электромагнитного излучения [9]. Особенно актуален данный вопрос для работников электроэнергетических предприятий, где уровень электромагнитного излучения максимальный.

Например, персонал, обслуживающий высоковольтную распределительную подстанцию. В распределительных устройствах, как открытого, так и закрытого типа, уровень электромагнитного излучения очень большой. В электроустановках 110кВ и выше, очень часто уровень электромагнитного излучения достигает таких значений, что его негативное воздействие на организм человека является очень сильным.

Первые признаки появляются практически сразу: головная боль, слабость, раздражительность, угнетенность [9]. В таких случаях нахождение человека в зоне действия электромагнитного излучения без использования специальных защитных комплектов (экранирующих устройств) недопустимо.

В таком случае следует, при наличии возможности, делать перерывы и выходить из помещения, тем самым сокращая время пребывания в зоне электромагнитного излучения. Также не лишним будет использовать вышеупомянутые устройства, которые позволяют минимизировать негативное воздействие электромагнитного излучения на организм человека.

Также следует отметить, что степень влияния электромагнитного излучения на организм человека напрямую зависит не только от времени пребывания в зоне его действия, но и от расстояния до источника излучения. То есть в процессе использования того или иного электроприбора или электрического устройства следует по возможности увеличивать расстояние до источника.

Например, при работе за компьютером рекомендуется ставить монитор на расстоянии не ближе 30 см от головы. То же самое касается телевизора и различных гаджетов.

При разговоре по мобильному телефону рекомендуется использовать громкую связь или проводную гарнитуру. Если мобильный телефон в данный момент не используется, не нужно его держать в кармане, лучше положить его на стол.

Как правило, в инструкции к электроприборам должны быть указаны меры безопасности, в частности безопасное расстояние к данному электроприбору, при котором уровень излучения будет минимальным. Если такие данные отсутствуют, то для своей же безопасности лучше эти данные уточнить. В интернете в свободном доступе есть информация по этому поводу.

Очень часто, как в быту, так и на работе, включены в сеть электроприборы, которые в данный момент не используются. К таким электроприборам можно отнести зарядные устройства мобильных телефонов, аудиоаппаратуру, видеоаппаратуру, телевизор и др. Отключение данных электроприборов позволяет значительно снизить уровень электромагнитного излучения и соответственно степень его негативного воздействия. Кроме того, отключение электроприборов позволяет снизить общее количество потребляемой электроэнергии.

# **7 РЕСУРСО- И ЭНЕРГОСБЕРЕЖЕНИЕ**

Производство любой продукции связано с потреблением сырья, энергии, использованием оборудования, интеллектуального и физического труда человека. Целесообразное использование ресурсов в производстве залог успешного бизнеса. Халатное отношение к ресурсам различных видов приводит к финансовым затратам, а также не отвечает современным тенденциям.

В Республике Беларусь в области ресурсо и энергосбережения действует Межгосударственный стандарт разработанный Межгосударственным Техническим комитетом по стандартизации МТК 111.

Стандарт «Ресурсосбережение» (ГОСТ 30166-95) устанавливает цель, задачи, объекты, основные принципы, термины и классификацию групп требований рационального использования и экономного расходования материальных ресурсов на всех стадиях жизненного цикла веществ, материалов, изделий, продукции при проведении работ и оказании услуг юридическим и физическим лицам. Настоящий стандарт распространяется на все виды деятельности, связанные с добычей, переработкой, транспортированием, хранением, распределением и потреблением материальных ресурсов, объектов.

Целью стандартизации в области ресурсосбережения является создание организационно-методической и нормативной основы, необходимой и достаточной для проведения государственной технической политики, направленной на снижение ресурсоемкости получаемого дохода без ухудшения условий экономического развития страны при безусловном обеспечении высоких потребительских свойств продукции.

Согласно стандарту «Ресурсосбережение» ресурсосбережение – деятельность (организационная, экономическая, техническая, научная, практическая, информационная), методы, процессы, комплекс организационно­ технических мер и мероприятий, сопровождающих все стадии жизненного цикла объектов и направленных на рациональное использование и экономное расходование ресурсов [10].

Стандарт «Энергетическая эффективность» устанавливает основные виды показателей энергосбережения и энергетической эффективности, вносимых в нормативные (технические, методические) документы, техническую (проектную, конструкторскую, технологическую, эксплуатационную) документацию на энергопотребляющую продукцию, технологические процессы, работы и услуги. Настоящий стандарт вводит определение слово энергосбережение – реализация правовых, организационных, научных, производственных, технических и экономических мер, направленных на эффективное (рациональное) использование (и экономное расходование) топливно-энергетических ресурсов и на вовлечение в хозяйственный оборот возобновляемых источников энергии [11].

Ресурсо- и энергосберегающие технологии использования средств вычислительной техники. В начале 90-х годов компания EPA (Environmental Protection Agency – Агентство по защите окружающей среды) начало проводить кампанию по сертификации энергосберегающих персональных компьютеров и периферийного оборудования. Компьютер или монитор во время продолжительного простоя должен снизить энергопотребление до 30 Вт и более. Система, удовлетворяющая этим требованиям, может получить сертификат Energy Star. В настоящее время в ПК нашли применение следующие энергосберегающие технологии.

Стандарт усовершенствованной системы управления питанием (Advanced Power Management – APM) разработан фирмой Intel совместно с Microsoft и определяет ряд интерфейсов между аппаратными средствами управления питанием и операционной системой компьютера. Полностью реализованный стандарт APM позволяет автоматически переключать компьютер между пятью состояниями в зависимости от текущего состояния системы. Каждое последующее состояние характеризуется уменьшением потребления энергии.

Full On: система полностью включена.

APM Enabled: система работает, некоторые устройства являются объектами управления для системы управления питанием. Неиспользуемые устройства могут быть выключены, может быть также остановлена или замедлена (т.е. снижена тактовая частота) работа тактового генератора центрального процессора.

APM Standby (резервный режим): система не работает, большинство устройств находятся в состоянии потребления малой мощности. Работа тактового генератора центрального процессора может быть замедлена или остановлена, но необходимые параметры функционирования хранятся в памяти. Пользователь или операционная система могут запустить компьютер из этого состояния почти мгновенно.

APM Suspend (режим приостановки): система не работает, большинство устройств пассивны. Тактовый генератор центрального процессора остановлен, а параметры функционирования хранятся на диске и при необходимости могут быть считаны в память для восстановления работы системы. Чтобы запустить систему из этого состояния, требуется некоторое время.

Off (система отключена): система не работает, источник питания выключен.

Усовершенствованная конфигурация и интерфейс питания (Advanced Configuration and Power Interface – ACPI) впервые реализованы в современных BIOS и операционных системах Windows 98 и более поздних. Если BIOS компьютера поддерживает систему ACPI, то все управление питанием передается операционной системе. Это упрощает конфигурирование параметров, все они находятся в одном месте – в операционной системе. Теперь для конфигурирования параметров системы управления питанием не нужно устанавливать соответствующие параметры в BIOS. Система ACPI реализована только в самых новых компьютерах.

Стандарт DPMS (Display Power Management Signaling – система сигналов управления питанием монитора) ассоциации VESA определяет состав сигналов, передаваемых компьютером в монитор, при вхождении системы от состояния простоя в режимы пониженного потребления энергии. В этих системных процедурах контроль берет на себя драйвер, посылающий соответствующие сигналы через графическую карту. При нажатии клавиши на клавиатуре или движении мыши монитор переходит в нормальный режим работы [12].

Вопрос экономии ресурсов и электроэнергии требует комплексного подхода, чтобы максимально эффективно использовать все производственные мощности при минимально возможных энергетических затратах. Подход к экономии электроэнергии основан на использовании энергосберегающих технологий, которые призваны уменьшить потери электроэнергии.

# **ЗАКЛЮЧЕНИЕ**

В ходе дипломной работы был разработан программный комплекс по подбору лекарственных средств на основе экспертных знаний, который предназначен для использования в повседневной жизни. Также был проведен анализ существующих приложений, предоставляющих поиск и подбор лекарственных средств.

На основании поставленной задачи было выполнено следующее:

* разработана структура приложения;
* спроектирована и разработана база данных;
* разработано серверное приложение, имеющее доступ к базе данных и предоставляющее определенный интерфейс для взаимодействия с ним;
* разработано клиентское приложение с применением самых современных и актуальных технологий;
* проведено функциональное тестирование важнейших функций приложения.

Разработанная система решила следующие задачи:

* подбор лекарственных средств и лечения;
* оценка пользователем полезности предоставленной информации о заболевании и методах их лечения;
* создана система рейтингов для заболеваний, на основе оценок пользователей;
* анализ экспертами рейтинга заболевания;
* формирование базы знаний.

В результате выполнения дипломной работы был получен работающий, готовый к использованию программный комплекс. Была проведена верификация полученного приложения, что дает основание утверждать, что все требования к продукту были реализованы и работают исправно. Цели дипломной работы были выполнены в полной мере.

# 

# **Список использованных источников**

1. Node.js – [Электронный ресурс]. – Режим доступа: <https://metanit.com/web/nodejs/1.1.php> , – Дата доступа: 19.04.2020.
2. Sequelize – [Электронный ресурс]. – Режим доступа: http://docs.sequelizejs.com/ , – Дата доступа: 15.04.2020.
3. [СовременныйучебникJavaScript](https://learn.javascript.ru/) – [Электронный ресурс]. – Режим доступа: <https://learn.javascript.ru/> , – Дата доступа: 19.04.2020.
4. React – [Электронный ресурс]. – Режим доступа: http://docs.reactjs.com/ , – Дата доступа: 15.04.2020.
5. Кожевников, Е.А., Ермалинская Н.В. – Расчет экономической эффективности разработки программных продуктов: Методические указания по подготовке организационно-экономического раздела дипломных работ для студентов специальности 1-40 01 02 «Информационные системы и технологии (по направлениям)» дневной формы обучения. / Гомель: ГГТУ им. П. О. Сухого, 2012 – 68 с.
6. Гармаза, А. К. Охрана труда: учебное пособие для студентов высших учебных заведений по специальностям лесного профиля / А. К. Гармаза, И. Т. Ермак, Б. Р. Ладик. – Минск: БГТУ, 2010. – 366 с.
7. Электронный журнал по охране труда – [Электронный ресурс]. – Режим доступа: <http://www.helper.by/sredstva-zashiti-rabotayushix-obshie-polojeniya.html> , – Дата доступа: 30.05.2020.
8. Охрана труда. Портал для инженеров по охране труда Беларуси – [Электронный ресурс]. – Режим доступа: <https://ohranatruda.of.by/v-chem-sostoit-vrednoe-vliyanie-na-organizm-cheloveka-elektromagnitnyx-polej.html> , – Дата доступа: 01.06.2020.
9. Защита от электромагнитных излучений – [Электронный ресурс]. – Режим доступа: <https://www.asutpp.ru/kak-zaschititsya-ot-elektromagnitnogo-izlucheniya.html> , – Дата доступа: 01.06.2020.
10. Межгосударственный совет по стандартизации, методологии и сертификации, Межгосударственной стандарт. Ресурсосбережение. Основные положения – Москва: ИПК Издательство стандартов, 2001. – 5 с.
11. Межгосударственный совет по стандартизации, методологии и сертификации, Межгосударственной стандарт. Энергетическая эффективность – Москва: ФГУП «Стандартинформ», 2013. – 7 с.
12. StudFiles. Модернизация средств вычислительной техники с учетом решаемых задач. – Энергосберегающие технологии [Электронный ресурс]. – Режим доступа: https://studfiles.net/preview/3000237/page:3/ , – Дата доступа: 03.06.2020.

ПРИЛОЖЕНИЕ А

(обязательное)

**Исходный код приложения**

import { createContainer, asFunction, asValue } from 'awilix';

import { app, methods, IMethods } from './app';

import { config } from './config';

import { database } from './infra/database';

import { repository, IRepository } from './infra/repositories';

import { constants, IConstants } from './infra/constants';

import { server } from './interfaces/http';

export interface IContainer {

  repository: IRepository;

  methods: IMethods; // eslint-disable-line @typescript-eslint/no-explicit-any

  server: any; // eslint-disable-line @typescript-eslint/no-explicit-any

  database: any; // eslint-disable-line @typescript-eslint/no-explicit-any

  config: any; // eslint-disable-line @typescript-eslint/no-explicit-any

  constants: IConstants;

}

export const container = createContainer();

container.register({

  app: asFunction(app).singleton(),

  methods: asFunction(methods).singleton(),

  server: asFunction(server).singleton(),

  database: asFunction(database).singleton(),

  repository: asFunction(repository).singleton(),

  config: asValue(config),

  constants: asValue(constants),

});

import express from 'express';

import session from 'express-session';

import bodyParser from 'body-parser';

import path from 'path';

import { IServer } from './interfaces';

import { router } from './router';

export const server = ({ config: { PORT, SESSION } }): IServer => {

  const app = express();

  app.use(bodyParser.json());

  app.use(session(SESSION));

  app.use('/api', router());

  app.use(express.static(path.join(\_\_dirname, '../../../../pill.client/build')));

  app.get('/', (\_, res) => res.send('May the force be with you!!!'));

  return {

    start: () => app.listen(PORT, () => console.log(`Api was started on ${PORT}`)),

  };

};

import { Router } from 'express';

import authRouter from './routes/v1/auth';

import userRouter from './routes/v1/user';

import illnessRouter from './routes/v1/illness';

import medicineRouter from './routes/v1/medicine';

import symptomRouter from './routes/v1/symptom';

import recommendationRouter from './routes/v1/recommendation';

import illnessReviewRouter from './routes/v1/illnessReview';

export const router = () => Router()

  .use('/v1/auth', authRouter())

  .use('/v1/user', userRouter())

  .use('/v1/illness', illnessRouter())

  .use('/v1/medicine', medicineRouter())

  .use('/v1/symptom', symptomRouter())

  .use('/v1/recommendation', recommendationRouter())

  .use('/v1/illness-review', illnessReviewRouter());

import { Request, Response, NextFunction } from 'express';

import { UNAUTHORIZED, FORBIDDEN } from 'http-status';

import { ROLE } from '../../../infra/constants';

export const isAuthorized = (req: Request, res: Response, next: NextFunction) => {

  if (req.session.profile) {

    next();

  } else {

    res.status(UNAUTHORIZED).send();

  }

};

export const isAdmin = (req: Request, res: Response, next: NextFunction) => {

  if (req.session.profile.role === ROLE.ADMIN) {

    next();

  } else {

    res.status(FORBIDDEN).send();

  }

};

import router from './router';

import { container, IContainer } from '../../../../../container';

export default () => {

  const {

    methods: { user: { Get, Post } },

    repository: { userRepository },

  }: IContainer = container.cradle;

  const getUseCase = new Get(userRepository);

  const createUseCase = new Post(userRepository);

  return router(getUseCase, createUseCase);

};

import { Router } from 'express';

import { BAD\_REQUEST } from 'http-status';

import { Get } from '../../../../../app/user';

export default (getUseCase: Get) => {

  const router = Router();

  router.post('/login', (req, res) => getUseCase

    .byCredentials(req.body.email, req.body.password)

    .then((data) => {

      if (data) {

        req.session.profile = data;

        res.send();

      } else {

        res.status(BAD\_REQUEST).send();

      }

    }),

  );

  router.post('/registration', (req, res) => createUseCase

    .create(req.body)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  return router;

};

import router from './router';

import { container, IContainer } from '../../../../../container';

export default () => {

  const {

    methods: { illness: { Get, Post, Delete } },

    repository: { illnessRepository },

  }: IContainer = container.cradle;

  const getUseCase = new Get(illnessRepository);

  const createUseCase = new Post(illnessRepository);

  const deleteUseCase = new Delete(illnessRepository);

  return router(getUseCase, createUseCase, deleteUseCase);

};

import { Router } from 'express';

import { BAD\_REQUEST, NOT\_FOUND } from 'http-status';

import { isAuthorized, isAdmin } from '../../../middlewares';

import { Get, Post, Delete } from '../../../../../app/illness';

export default (

  getUseCase: Get,

  createUseCase: Post,

  deleteUseCase: Delete,

) => {

  const router = Router();

  router.get('/', isAuthorized, (req, res) => getUseCase

    .all(req.query)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.post('/', isAuthorized, isAdmin, (req, res) => createUseCase

    .create(req.body)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.get('/short', isAuthorized, (req, res) => getUseCase

    .allShort()

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.get('/symptoms', isAuthorized, (req, res) => getUseCase

    .bySymptomIds(req.query.ids)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.get('/:id', isAuthorized, (req, res) => getUseCase

    .byId(Number(req.params.id))

    .then(data => res.send(data))

    // .catch(error => res.status(NOT\_FOUND).send(error)),

  );

  router.delete('/:id', isAuthorized, isAdmin, (req, res) => deleteUseCase

    .delete(Number(req.params.id))

    .then(() => res.send())

    .catch(error => res.status(NOT\_FOUND).send(error)),

  );

  return router;

};

import router from './router';

import { container, IContainer } from '../../../../../container';

export default () => {

  const {

    methods: { illnessReview: { Get, Post } },

    repository: { illnessReviewRepository },

  }: IContainer = container.cradle;

  const getUseCase = new Get(illnessReviewRepository);

  const createUseCase = new Post(illnessReviewRepository);

  return router(getUseCase, createUseCase);

};

import { Router } from 'express';

import { BAD\_REQUEST } from 'http-status';

import { isAuthorized, isAdmin } from '../../../middlewares';

import { Get, Post } from '../../../../../app/illnessReview';

export default (

  getUseCase: Get,

  createUseCase: Post,

) => {

  const router = Router();

  router.get('/', isAuthorized, isAdmin, (req, res) => getUseCase

    .all(req.body)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.post('/', isAuthorized, (req, res) => createUseCase

    .create(req.body)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.get('/:id', isAuthorized, isAdmin, (req, res) => getUseCase

    .allByIllnessId(Number(req.params.id))

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.get('/graphic/:id', isAuthorized, isAdmin, (req, res) => getUseCase

    .graphicById(Number(req.params.id))

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  return router;

};

import { Router } from 'express';

import { BAD\_REQUEST } from 'http-status';

import { isAuthorized, isAdmin } from '../../../middlewares';

import { Get, Post } from '../../../../../app/medicine';

export default (

  getUseCase: Get,

  createUseCase: Post,

) => {

  const router = Router();

  router.get('/', isAuthorized, (\_, res) => getUseCase

    .all()

    .then(data => res.send(data)),

  );

  router.post('/', isAuthorized, isAdmin, (req, res) => createUseCase

    .create(req.body)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.get('/conflicts', isAuthorized, (\_, res) => getUseCase

    .conflicts()

    .then(data => res.send(data)),

  );

  router.get('/illness/:id', isAuthorized, (req, res) => getUseCase

    .byIllnessId(req.params.id)

    .then(data => res.send(data)),

  );

  return router;

};

import router from './router';

import { container, IContainer } from '../../../../../container';

export default () => {

  const {

    methods: { recommendation: { Get } },

    repository: { recommendationRepository },

  }: IContainer = container.cradle;

  const getUseCase = new Get(recommendationRepository);

  return router(getUseCase);

};

import { Router } from 'express';

import { BAD\_REQUEST } from 'http-status';

import { isAuthorized } from '../../../middlewares';

import { Get } from '../../../../../app/recommendation';

export default (

  getUseCase: Get,

) => {

  const router = Router();

  router.get('/', isAuthorized, (\_, res) => getUseCase

    .all()

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  return router;

};

import router from './router';

import { container, IContainer } from '../../../../../container';

export default () => {

  const {

    methods: { symptom: { Get, Post, Put, Delete } },

    repository: { symptomRepository },

  }: IContainer = container.cradle;

  const getUseCase = new Get(symptomRepository);

  const createUseCase = new Post(symptomRepository);

  const updateUseCase = new Put(symptomRepository);

  const deleteUseCase = new Delete(symptomRepository);

  return router(getUseCase, createUseCase, updateUseCase, deleteUseCase);

};

import { Router } from 'express';

import { BAD\_REQUEST } from 'http-status';

import { isAuthorized } from '../../../middlewares';

import { Get, Post, Put, Delete } from '../../../../../app/symptom';

export default (getUseCase: Get, createUseCase: Post, updateUseCase: Put, deleteUseCase: Delete) => {

  const router = Router();

  router.get('/', isAuthorized, (req, res) => getUseCase

    .all(req.query)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.post('/', isAuthorized, (req, res) => createUseCase

    .create(req.body)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.put('/', isAuthorized, (req, res) => updateUseCase

    .update(req.body)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.get('/short', isAuthorized, (\_, res) => getUseCase

    .allShort()

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.delete('/:id', isAuthorized, (req, res) => deleteUseCase

    .byId(Number(req.params.id))

    .then(() => res.send())

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  return router;

};

import router from './router';

import { container, IContainer } from '../../../../../container';

export default () => {

  const {

    methods: { user: { Get, Post } },

    repository: { userRepository },

  }: IContainer = container.cradle;

  const getUseCase = new Get(userRepository);

  const createUseCase = new Post(userRepository);

  return router(getUseCase, createUseCase);

};

import { Router } from 'express';

import { BAD\_REQUEST } from 'http-status';

import { isAuthorized, isAdmin } from '../../../middlewares';

import { Get, Post } from '../../../../../app/user';

export default (

  getUseCase: Get,

  createUseCase: Post,

) => {

  const router = Router();

  router.get('/', isAuthorized, isAdmin, (\_, res) => getUseCase

    .all()

    .then(data => res.send(data)),

  );

  router.post('/', isAuthorized, isAdmin, (req, res) => createUseCase

    .create(req.body)

    .then(data => res.send(data))

    .catch(error => res.status(BAD\_REQUEST).send(error)),

  );

  router.get('/profile', isAuthorized, (req, res) => res.send(req.session.profile));

  router.get('/:id', isAuthorized, isAdmin, (req, res) => getUseCase

    .byId(Number(req.params.id))

    .then(data => res.send(data)),

  );

  return router;

};

export const ROLE = {

  USER: 1,

  ADMIN: 2,

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('Users', {

      id: {

        allowNull: false,

        autoIncrement: true,

        primaryKey: true,

        type: Sequelize.INTEGER,

      },

      firstName: {

        allowNull: false,

        type: Sequelize.STRING,

      },

      lastName: {

        allowNull: false,

        type: Sequelize.STRING,

      },

      email: {

        allowNull: false,

        unique: true,

        validate: {

          isEmail: true,

        },

        type: Sequelize.STRING,

      },

      password: {

        allowNull: false,

        type: Sequelize.STRING,

      },

      role: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('Users');

  },

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('Illnesses', {

      id: {

        allowNull: false,

        autoIncrement: true,

        primaryKey: true,

        type: Sequelize.INTEGER,

      },

      name: {

        allowNull: false,

        unique: true,

        type: Sequelize.STRING,

      },

      description: {

        type: Sequelize.TEXT,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('Illnes');

  },

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('Medicines', {

      id: {

        allowNull: false,

        autoIncrement: true,

        primaryKey: true,

        type: Sequelize.INTEGER,

      },

      name: {

        allowNull: false,

        unique: true,

        type: Sequelize.STRING,

      },

      description: {

        allowNull: false,

        type: Sequelize.TEXT,

      },

      wayUse: {

        allowNull: false,

        type: Sequelize.TEXT,

      },

      typeMedicineId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      routeId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      productionMedicineMethodId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('Medicines');

  },

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('Symptoms', {

      id: {

        allowNull: false,

        autoIncrement: true,

        primaryKey: true,

        type: Sequelize.INTEGER,

      },

      name: {

        allowNull: false,

        unique: true,

        type: Sequelize.STRING,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      }

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('Symptoms');

  }

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('IllnessSymptoms', {

      illnessId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      symptomId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('IllnessSymptoms');

  }

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('IllnessMedicines', {

      illnessId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      medicineId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('IllnessMedicines');

  }

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('Recommendations', {

      id: {

        allowNull: false,

        autoIncrement: true,

        primaryKey: true,

        type: Sequelize.INTEGER,

      },

      name: {

        allowNull: false,

        unique: true,

        type: Sequelize.STRING,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('Recommendations');

  },

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('IllnessRecommendations', {

      illnessId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      recommendationId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('IllnessRecommendations');

  },

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('IllnessReviews', {

      id: {

        allowNull: false,

        autoIncrement: true,

        primaryKey: true,

        type: Sequelize.INTEGER,

      },

      illnessId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      review: {

        allowNull: false,

        type: Sequelize.TEXT,

      },

      descriptionRating: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      symptomsRating: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      recommendationsRating: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('IllnessReviews');

  },

};

'use strict';

module.exports = {

  up: async (queryInterface, Sequelize) => {

    await queryInterface.createTable('IllnessReviewMedicines', {

      id: {

        allowNull: false,

        autoIncrement: true,

        primaryKey: true,

        type: Sequelize.INTEGER,

      },

      illnessReviewId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      medicineId: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      rating: {

        allowNull: false,

        type: Sequelize.INTEGER,

      },

      createdAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

      updatedAt: {

        allowNull: false,

        type: Sequelize.DATE,

      },

    });

  },

  down: async (queryInterface) => {

    await queryInterface.dropTable('IllnessReviewMedicines');

  },

};

import { Sequelize } from 'sequelize-typescript';

import { Dialect } from 'sequelize';

import path from 'path';

export const database = ({ config: { DB\_USERNAME, DB\_PASSWORD, DB\_NAME, DB\_HOST, DB\_DIALECT } }) => {

  const sequelize = new Sequelize({

    dialect: DB\_DIALECT as Dialect,

    database: DB\_NAME,

    username: DB\_USERNAME,

    password: DB\_PASSWORD,

    host: DB\_HOST,

    models: [path.resolve(\_\_dirname, '..', 'models')],

  });

  (sequelize as any).connect = () => sequelize // eslint-disable-line @typescript-eslint/no-explicit-any

    .authenticate()

    .then(() => console.log('Database was connected successfully'));

  return sequelize;

};

import {

  Model,

  Column,

  Table,

  PrimaryKey,

  CreatedAt,

  UpdatedAt,

  AllowNull,

  NotEmpty,

  AutoIncrement,

  Unique,

  BelongsToMany,

  HasMany,

} from 'sequelize-typescript';

import { IIllness } from '../../domain';

import { IllnessSymptom } from './IllnessSymptom';

import { IllnessMedicine } from './IllnessMedicine';

import { IllnessRecommendation } from './IllnessRecommendation';

import { Symptom } from './Symptom';

import { Medicine } from './Medicine';

import { Recommendation } from './Recommendation';

import { IllnessReview } from './IllnessReview';

@Table

export class Illness extends Model<Illness> implements IIllness {

  @PrimaryKey

  @AutoIncrement

  @Column

  public id: number;

  @AllowNull(false)

  @NotEmpty

  @Unique

  @Column

  public name: string;

  @AllowNull(false)

  @NotEmpty

  @Column

  public description: string;

  @BelongsToMany(() => Symptom, () => IllnessSymptom)

  public symptoms: Symptom[];

  @BelongsToMany(() => Medicine, () => IllnessMedicine)

  public medicines: Medicine[];

  @BelongsToMany(() => Recommendation, () => IllnessRecommendation)

  public recommendations: Recommendation[];

  @HasMany(() => IllnessReview)

  public reviews: IllnessReview[];

  @CreatedAt

  @Column

  public createdAt: Date;

  @UpdatedAt

  @Column

  public updatedAt: Date;

}

import {

  Model,

  Column,

  Table,

  ForeignKey,

} from 'sequelize-typescript';

import { Illness } from './Illness';

import { Medicine } from './Medicine';

@Table

export class IllnessMedicine extends Model<IllnessMedicine> {

  @ForeignKey(() => Illness)

  @Column

  public illnessId: number;

  @ForeignKey(() => Medicine)

  @Column

  public medicineId: number;

}

import {

  Model,

  Column,

  Table,

  ForeignKey,

} from 'sequelize-typescript';

import { Illness } from './Illness';

import { Recommendation } from './Recommendation';

@Table

export class IllnessRecommendation extends Model<IllnessRecommendation> {

  @ForeignKey(() => Illness)

  @Column

  public illnessId: number;

  @ForeignKey(() => Recommendation)

  @Column

  public recommendationId: number;

}

import {

  Model,

  Column,

  Table,

  PrimaryKey,

  CreatedAt,

  UpdatedAt,

  AllowNull,

  NotEmpty,

  AutoIncrement,

  ForeignKey,

  DataType,

  HasMany,

} from 'sequelize-typescript';

import { Illness } from './Illness';

import { IllnessReviewMedicine } from './IllnessReviewMedicine';

import { IIllnessReview } from '../../domain';

@Table

export class IllnessReview extends Model<IllnessReview> implements IIllnessReview {

  @PrimaryKey

  @AutoIncrement

  @Column

  public id: number;

  @AllowNull(false)

  @NotEmpty

  @ForeignKey(() => Illness)

  @Column

  public illnessId: number;

  @AllowNull(false)

  @Column({ type: DataType.TEXT })

  public review: string;

  @AllowNull(false)

  @Column

  public descriptionRating: number;

  @AllowNull(false)

  @Column

  public symptomsRating: number;

  @AllowNull(false)

  @Column

  public recommendationsRating: number;

  @HasMany(() => IllnessReviewMedicine)

  public medicines: IllnessReviewMedicine[];

  @CreatedAt

  @Column

  public createdAt: Date;

  @UpdatedAt

  @Column

  public updatedAt: Date;

}

import {

  Model,

  Column,

  Table,

  PrimaryKey,

  CreatedAt,

  UpdatedAt,

  AllowNull,

  NotEmpty,

  AutoIncrement,

  Unique,

  ForeignKey,

  DataType,

  BelongsTo,

} from 'sequelize-typescript';

import { IllnessReview } from './IllnessReview';

import { Medicine } from './Medicine';

import { IIllnessReviewMedicine } from '../../domain';

@Table

export class IllnessReviewMedicine extends Model<IllnessReviewMedicine> implements IIllnessReviewMedicine {

  @PrimaryKey

  @AutoIncrement

  @Column

  public id: number;

  @AllowNull(false)

  @NotEmpty

  @ForeignKey(() => IllnessReview)

  @Column

  public illnessReviewId: number;

  @AllowNull(false)

  @NotEmpty

  @ForeignKey(() => Medicine)

  @Column

  public medicineId: number;

  @BelongsTo(() => Medicine)

  public medicine: Medicine;

  @AllowNull(false)

  @Column

  public rating: number;

  @CreatedAt

  @Column

  public createdAt: Date;

  @UpdatedAt

  @Column

  public updatedAt: Date;

}

import {

  Model,

  Column,

  Table,

  ForeignKey,

} from 'sequelize-typescript';

import { Illness } from './Illness';

import { Symptom } from './Symptom';

@Table

export class IllnessSymptom extends Model<IllnessSymptom> {

  @ForeignKey(() => Illness)

  @Column

  public illnessId: number;

  @ForeignKey(() => Symptom)

  @Column

  public symptomId: number;

}

import {

  Model,

  Column,

  Table,

  PrimaryKey,

  CreatedAt,

  UpdatedAt,

  AllowNull,

  NotEmpty,

  AutoIncrement,

  Unique,

  ForeignKey,

  BelongsTo,

  DataType,

  BelongsToMany,

} from 'sequelize-typescript';

import { IMedicine } from '../../domain';

import { Route } from './Route';

import { TypeMedicine } from './TypeMedicine';

import { ProductionMedicineMethod } from './ProductionMedicineMethod';

import { Illness } from './Illness';

import { IllnessMedicine } from './IllnessMedicine';

@Table

export class Medicine extends Model<Medicine> implements IMedicine {

  @PrimaryKey

  @AutoIncrement

  @Column

  public id: number;

  @AllowNull(false)

  @NotEmpty

  @Unique

  @Column

  public name: string;

  @AllowNull(false)

  @NotEmpty

  @Column({ type: DataType.TEXT })

  public description: string;

  @AllowNull(false)

  @NotEmpty

  @Column({ type: DataType.TEXT })

  public wayUse: string;

  @AllowNull(false)

  @NotEmpty

  @ForeignKey(() => TypeMedicine)

  @Column

  public typeMedicineId: number;

  @BelongsTo(() => TypeMedicine)

  public typeMedicine: TypeMedicine;

  @AllowNull(false)

  @NotEmpty

  @ForeignKey(() => Route)

  @Column

  public routeId: number;

  @BelongsTo(() => Route)

  public route: Route;

  @AllowNull(false)

  @NotEmpty

  @ForeignKey(() => ProductionMedicineMethod)

  @Column

  public productionMedicineMethodId: number;

  @BelongsTo(() => ProductionMedicineMethod)

  public productionMedicineMethod: ProductionMedicineMethod;

  @BelongsToMany(() => Illness, () => IllnessMedicine)

  public illnesses: Illness[];

  @CreatedAt

  @Column

  public createdAt: Date;

  @UpdatedAt

  @Column

  public updatedAt: Date;

}

import {

  Model,

  Column,

  Table,

  PrimaryKey,

  CreatedAt,

  UpdatedAt,

  AllowNull,

  NotEmpty,

  AutoIncrement,

  Unique,

} from 'sequelize-typescript';

import { IRecommendation } from '../../domain';

@Table

export class Recommendation extends Model<Recommendation> implements IRecommendation {

  @PrimaryKey

  @AutoIncrement

  @Column

  public id: number;

  @AllowNull(false)

  @NotEmpty

  @Unique

  @Column

  public name: string;

  @CreatedAt

  @Column

  public createdAt: Date;

  @UpdatedAt

  @Column

  public updatedAt: Date;

}

import {

  Model,

  Column,

  Table,

  PrimaryKey,

  CreatedAt,

  UpdatedAt,

  AllowNull,

  NotEmpty,

  AutoIncrement,

  Unique,

} from 'sequelize-typescript';

import { IRoute } from '../../domain';

@Table

export class Route extends Model<Route> implements IRoute {

  @PrimaryKey

  @AutoIncrement

  @Column

  public id: number;

  @AllowNull(false)

  @NotEmpty

  @Unique

  @Column

  public name: string;

  @CreatedAt

  @Column

  public createdAt: Date;

  @UpdatedAt

  @Column

  public updatedAt: Date;

}

import {

  Model,

  Column,

  Table,

  PrimaryKey,

  CreatedAt,

  UpdatedAt,

  AllowNull,

  NotEmpty,

  AutoIncrement,

  Unique,

  BelongsToMany,

} from 'sequelize-typescript';

import { ISymptom } from '../../domain';

import { Illness } from './Illness';

import { IllnessSymptom } from './IllnessSymptom';

@Table

export class Symptom extends Model<Symptom> implements ISymptom {

  @PrimaryKey

  @AutoIncrement

  @Column

  public id: number;

  @AllowNull(false)

  @NotEmpty

  @Unique

  @Column

  public name: string;

  @BelongsToMany(() => Illness,() => IllnessSymptom)

  public illnesses;

  @CreatedAt

  @Column

  public createdAt: Date;

  @UpdatedAt

  @Column

  public updatedAt: Date;

}

import {

  Model,

  Column,

  Table,

  PrimaryKey,

  CreatedAt,

  UpdatedAt,

  AllowNull,

  NotEmpty,

  IsEmail,

  AutoIncrement,

  Is,

} from 'sequelize-typescript';

import { IUser } from '../../domain';

import { ROLE } from '../constants';

@Table

export class User extends Model<User> implements IUser {

  @PrimaryKey

  @AutoIncrement

  @Column

  public id: number;

  @AllowNull(false)

  @NotEmpty

  @Column

  public firstName: string;

  @AllowNull(false)

  @NotEmpty

  @Column

  public lastName: string;

  @AllowNull(false)

  @NotEmpty

  @IsEmail

  @Column

  public email: string;

  @AllowNull(false)

  @NotEmpty

  @Column

  public password: string;

  @AllowNull(false)

  @NotEmpty

  @Is('Role', (value) => {

    if (!Object.values(ROLE).includes(value)) {

      throw new Error(`${value} is not a existed role`);

    }

  })

  @Column

  public role: number;

  @CreatedAt

  @Column

  public createdAt: Date;

  @UpdatedAt

  @Column

  public updatedAt: Date;

}

import { DATE\_ATTRIBUTES\_EXCLUDE } from './constants';

import { ICreateIllness } from '../../domain';

import { Illness } from '../models/Illness';

import { Symptom } from '../models/Symptom';

import { Medicine } from '../models/Medicine';

import { Recommendation } from '../models/Recommendation';

import { IllnessSymptom } from '../models/IllnessSymptom';

import { IllnessMedicine } from '../models/IllnessMedicine';

import { IllnessRecommendation } from '../models/IllnessRecommendation';

import { IllnessReview } from '../models/IllnessReview';

import { IllnessReviewMedicine } from '../models/IllnessReviewMedicine';

import { getAverageRating } from '../utils';

export class IllnessRepository {

  public getAll(query: object) {

    return Illness.findAll({ where: query });

  }

  public getAllShort() {

    return Illness.findAll({

      attributes: {

        exclude: ['description'],

      },

    });

  }

  public getById(id: number) {

    return Illness.findOne({

      where: { id },

      include: [

        {

          model: Symptom,

          attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

          through: { attributes: [] },

        },

        {

          model: Recommendation,

          attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

          through: { attributes: [] },

        },

        {

          model: Medicine,

          attributes: { exclude: [...DATE\_ATTRIBUTES\_EXCLUDE, 'typeMedicineId', 'routeId', 'productionMedicineMethodId'] },

          through: { attributes: [] },

        },

        {

          model: IllnessReview,

          include: [{

            model: IllnessReviewMedicine,

          }],

        },

      ],

    });

  }

  public async getBySymptomIds(ids: number[]) {

    const illnesses = await Illness.findAll({

      attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

      include: [

        {

          model: Symptom,

          where: { id: ids },

          attributes: [],

        },

        {

          model: IllnessReview,

          include: [{

            model: IllnessReviewMedicine,

          }],

        },

      ],

    });

    return illnesses.map(({ id, name, reviews }) => ({

      id,

      name,

      rating: getAverageRating(reviews),

    }));

  }

  public async create(illness: ICreateIllness) {

    const { id: illnessId } = await Illness.create(illness);

    await Promise.all([

      IllnessSymptom.bulkCreate(illness.symptomsIds.map(symptomId => ({ symptomId, illnessId }))),

      IllnessMedicine.bulkCreate(illness.medicinesIds.map(medicineId => ({ medicineId, illnessId }))),

      IllnessRecommendation.bulkCreate(illness.recommendationsIds.map(recommendationId => ({ recommendationId, illnessId }))),

    ]);

  }

  public delete(id: number) {

    return Illness.destroy({ where: { id } });

  }

}

import { IllnessReview } from '../models/IllnessReview';

import { IllnessReviewMedicine } from '../models/IllnessReviewMedicine';

import { Illness } from '../models/Illness';

import { Medicine } from '../models/Medicine';

import { getAverageIllnessRating, getAverageMedicinesRating } from '../utils';

import { IIllnessReview } from '../../domain';

export class IllnessReviewRepository {

  public getAll(query: object) {

    return IllnessReview.findAll({ where: query });

  }

  public getAllByIllnessId(illnessId: number) {

    return IllnessReview.findAll({

      where: { illnessId },

      include: {

        model: IllnessReviewMedicine,

        include: {

          model: Medicine,

          attributes: ['name'],

        },

      },

    });

  }

  public async getGraphicById(id: number) {

    const illnessReview = await Illness.findOne({

      where: { id },

      attributes: ['id', 'name'],

      include: {

        model: IllnessReview,

        include: {

          model: IllnessReviewMedicine,

          include: {

            model: Medicine,

            attributes: ['name'],

          },

        },

      },

    });

    const allMedicines = illnessReview.reviews.reduce((prev, { medicines }) => [...prev, ...medicines], []);

    const reviews = getAverageIllnessRating(illnessReview.reviews);

    const medicines = getAverageMedicinesRating(allMedicines);

    return {

      name: illnessReview.name,

      reviews,

      medicines,

    };

  }

  public async create(illnessReview: IIllnessReview) {

    const { id: illnessReviewId } = await IllnessReview.create(illnessReview);

    await IllnessReviewMedicine.bulkCreate(illnessReview.medicines.map(m => ({ ...m, illnessReviewId })));

  }

}

import { DATE\_ATTRIBUTES\_EXCLUDE } from './constants';

import { getMedicineConflicts, getGroupConflicts } from '../utils';

import { IMedicineConflicts } from '../interfaces';

import { IMedicine } from '../../domain';

import { Medicine } from '../models/Medicine';

import { Route } from '../models/Route';

import { TypeMedicine } from '../models/TypeMedicine';

import { ProductionMedicineMethod } from '../models/ProductionMedicineMethod';

import { MedicineConflict } from '../models/MedicineConflict';

import { MedicineGroup } from '../models/MedicineGroup';

import { MedicineGroupConflict } from '../models/MedicineGroupConflict';

import { Illness } from '../models/Illness';

export class MedicineRepository {

  public getAll() {

    return Medicine.findAll({

      attributes: { exclude: ['typeMedicineId', 'routeId', 'productionMedicineMethodId'] },

      include: [

        {

          model: Route,

          attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

        },

        {

          model: TypeMedicine,

          attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

        },

        {

          model: ProductionMedicineMethod,

          attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

        },

      ],

    });

  }

  public getAllByIllnessId(illnessId: number) {

    return Medicine.findAll({

      attributes: ['id', 'name'],

      include: {

        model: Illness,

        where: { id: illnessId },

        attributes: [],

      },

    });

  }

  public create(medicine: IMedicine) {

    return Medicine.create(medicine);

  }

  public async getMedicineConflicts(medicineIds: number[]): Promise<IMedicineConflicts> {

    const [medicineConflictsArr, medicineGroupsArr, medicineGroupConflictsArr] = await Promise.all([

      MedicineConflict.findAll({

        attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

        where: {

          medicineId: medicineIds,

          conflictMedicineId: medicineIds,

        },

      }),

      MedicineGroup.findAll({

        attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

        where: { medicineId: medicineIds },

      }),

      MedicineGroupConflict.findAll({

        attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

        where: { medicineId: medicineIds },

      }),

    ]);

    const medicineConflicts = getMedicineConflicts(medicineConflictsArr);

    const groupConflicts = getGroupConflicts(medicineGroupsArr, medicineGroupConflictsArr, medicineConflicts);

    return {

      medicineConflicts,

      groupConflicts,

    };

  }

}

import { DATE\_ATTRIBUTES\_EXCLUDE } from './constants';

import { Recommendation } from '../models/Recommendation';

export class RecommendationRepository {

  public getAll() {

    return Recommendation.findAll({

      attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

    });

  }

}

import { Symptom } from '../models/Symptom';

import { DATE\_ATTRIBUTES\_EXCLUDE } from './constants';

import { ISymptom } from '../../domain';

export class SymptomRepository {

  public getAll(query) {

    return Symptom.findAll({ where: query });

  }

  public getAllShort() {

    return Symptom.findAll({

      attributes: { exclude: DATE\_ATTRIBUTES\_EXCLUDE },

    });

  }

  public create(symptom: ISymptom) {

    return Symptom.create(symptom);

  }

  public async update({ id, ...params }: ISymptom) {

    const symptom = await Symptom.findOne({ where: { id } });

    return await symptom.update(params);

  }

  public delete(id: number) {

    return Symptom.destroy( { where: { id } });

  }

}

import { User } from '../models/User';

import { IUser } from '../../domain';

export class UserRepository {

  public getAll(...args) {

    return User.findAll(...args);

  }

  public getOne(...args) {

    return User.findOne(...args);

  }

  public create(user: IUser) {

    return User.create(user);

  }

  public remove(...args) {

    return User.destroy(...args);

  }

}

import { IllnessRepository } from '../../infra/repositories';

export class Delete {

  constructor(

    private illnessRepository: IllnessRepository,

  ) { }

  public delete(id: number) {

    return this.illnessRepository.delete(id);

  }

}

import { IllnessRepository } from '../../infra/repositories';

export class Get {

  public constructor(

    private illnessRepository: IllnessRepository,

  ) { }

  public all(query: object) {

    return this.illnessRepository.getAll(query);

  }

  public allShort() {

    return this.illnessRepository.getAllShort();

  }

  public byId(id: number) {

    return this.illnessRepository.getById(id);

  }

  public bySymptomIds(ids: number[]) {

    if (!ids || !ids.length) {

      return Promise.reject();

    }

    return this.illnessRepository.getBySymptomIds(ids);

  }

}

import { PathReporter } from 'io-ts/lib/PathReporter';

import { isLeft } from 'fp-ts/lib/Either';

import { IIllness, CreateIllness } from '../../domain';

import { IllnessRepository } from '../../infra/repositories';

export class Post {

  public constructor(

    private illnessRepository: IllnessRepository,

  ) { }

  public create(illness: IIllness) {

    const entity = CreateIllness.decode(illness);

    if (isLeft(entity)) {

      return Promise.reject(PathReporter.report(entity));

    }

    return this.illnessRepository.create(illness);

  }

}

import { IIllnessReview, IllnessReview} from '../../domain';

import { IllnessReviewRepository } from '../../infra/repositories';

export class Get {

  constructor(

    private illnessReviewRepository: IllnessReviewRepository,

  ) { }

  public all(query: object) {

    return this.illnessReviewRepository.getAll(query);

  }

  public allByIllnessId(illnessId: number) {

    return this.illnessReviewRepository.getAllByIllnessId(illnessId);

  }

  public graphicById(id: number) {

    return this.illnessReviewRepository.getGraphicById(id);

  }

}

import { PathReporter } from 'io-ts/lib/PathReporter';

import { isLeft } from 'fp-ts/lib/Either';

import { IIllnessReview, IllnessReview} from '../../domain';

import { IllnessReviewRepository } from '../../infra/repositories';

export class Post {

  constructor(

    private illnessReviewRepository: IllnessReviewRepository,

  ) { }

  public create(illnessReview: IIllnessReview) {

    const entity = IllnessReview.decode(illnessReview);

    if (isLeft(entity)) {

      return Promise.reject(PathReporter.report(entity));

    }

    return this.illnessReviewRepository.create(illnessReview);

  }

}

import { MedicineRepository } from '../../infra/repositories';

export class Get {

  public constructor(

    private medicineRepository: MedicineRepository,

  ) { }

  public all() {

    return this.medicineRepository.getAll();

  }

  public byIllnessId(illnessId: number) {

    return this.medicineRepository.getAllByIllnessId(illnessId);

  }

  public conflicts() {

    return this.medicineRepository.getMedicineConflicts([2, 4]);

  }

}

import { PathReporter } from 'io-ts/lib/PathReporter';

import { isLeft } from 'fp-ts/lib/Either';

import { Medicine, IMedicine } from '../../domain';

import { MedicineRepository } from '../../infra/repositories';

export class Post {

  public constructor(

    private medicineRepository: MedicineRepository,

  ) { }

  public create(medicine: IMedicine) {

    const entity = Medicine.decode(medicine);

    if (isLeft(entity)) {

      return Promise.reject(PathReporter.report(entity));

    }

    return this.medicineRepository.create(medicine);

  }

}

import { RecommendationRepository } from '../../infra/repositories';

export class Get {

  constructor(

    private recommendationRepository: RecommendationRepository,

  ) { }

  public all() {

    return this.recommendationRepository.getAll();

  }

}

import { SymptomRepository } from '../../infra/repositories';

export class Delete {

  constructor (

    private symptomRepository: SymptomRepository,

  ) { }

  public byId(id: number) {

    return this.symptomRepository.delete(id);

  }

}

import { SymptomRepository } from '../../infra/repositories';

export class Get {

  public constructor(

    private symptomRepository: SymptomRepository,

  ) { }

  public all(query: object) {

    return this.symptomRepository.getAll(query);

  }

  public allShort() {

    return this.symptomRepository.getAllShort();

  }

}

import { PathReporter } from 'io-ts/lib/PathReporter';

import { isLeft } from 'fp-ts/lib/Either';

import { SymptomRepository } from '../../infra/repositories';

import { Symptom, ISymptom } from '../../domain';

export class Post {

  public constructor(

    private symptomRepository: SymptomRepository,

  ) { }

  public create(symptom: ISymptom) {

    const entity = Symptom.decode(symptom);

    if (isLeft(entity)) {

      return Promise.reject(PathReporter.report(entity));

    }

    return this.symptomRepository.create(symptom);

  }

}

import { PathReporter } from 'io-ts/lib/PathReporter';

import { isLeft } from 'fp-ts/lib/Either';

import { SymptomRepository } from '../../infra/repositories';

import { Symptom, ISymptom } from '../../domain';

export class Put {

  constructor(

    private symptomRepository: SymptomRepository,

  ) { }

  public update(symptom: ISymptom) {

    const entity = Symptom.decode(symptom);

    if (isLeft(entity)) {

      return Promise.reject(PathReporter.report(entity));

    }

    return this.symptomRepository.update(symptom);

  }

}

import { container } from './src/container';

import { IApp } from './src/app';

container

    .resolve<IApp>('app')

    .start();

{

  "name": "pill.api",

  "version": "1.0.0",

  "main": "index.js",

  "author": "Uladzislau Pinchuk",

  "license": "MIT",

  "scripts": {

    "dev": "ts-node-dev --respawn --transpileOnly index.ts",

    "build": "rimraf build && tsc",

    "start": "node ./build/index.js",

    "lint": "eslint --ext .ts .",

    "db:create": "sequelize db:create",

    "db:drop": "sequelize db:drop",

    "db:migrate": "sequelize db:migrate",

    "db:migrate:undo": "sequelize db:migrate:undo",

    "db:migrate:undo:all": "sequelize db:migrate:undo:all",

    "db:seed": "sequelize db:seed:all",

    "db:refresh": "yarn run db:drop && yarn run db:create && yarn run db:migrate && yarn run db:seed"

  },

  "husky": {

    "hooks": {

      "pre-push": "",

      "commit-msg": "commitlint -E HUSKY\_GIT\_PARAMS"

    }

  },

  "dependencies": {

    "awilix": "4.2.4",

    "body-parser": "1.19.0",

    "express": "4.17.1",

    "express-session": "1.17.1",

    "fp-ts": "2.5.3",

    "http-status": "1.4.2",

    "husky": "4.2.1",

    "io-ts": "2.1.2",

    "mysql2": "2.1.0",

    "reflect-metadata": "0.1.13",

    "sequelize": "5.21.4",

    "sequelize-cli": "5.5.1"

  },

  "devDependencies": {

    "@commitlint/cli": "8.3.5",

    "@commitlint/config-conventional": "8.3.4",

    "@types/bluebird": "3.5.29",

    "@types/express": "4.17.2",

    "@types/express-session": "1.17.0",

    "@types/node": "13.7.0",

    "@types/validator": "12.0.1",

    "@typescript-eslint/eslint-plugin": "2.19.2",

    "@typescript-eslint/parser": "2.19.2",

    "dotenv": "8.2.0",

    "eslint": "6.8.0",

    "sequelize-typescript": "1.1.0",

    "ts-node-dev": "1.0.0-pre.44",

    "typescript": "3.7.5"

  }

}

import React from 'react';

import ReactDOM from 'react-dom';

import { AppContainer } from './components/App';

ReactDOM.render(

  <AppContainer />,

  document.getElementById('app'),

);

<!DOCTYPE html>

<html lang="en">

<head>

  <meta charset="UTF-8">

  <title>HM\_PLL</title>

  <meta name="viewport" content="minimum-scale=1, initial-scale=1, width=device-width" />

  <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Roboto:300,400,500,700&display=swap" />

  <link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons" />

</head>

<body>

  <style>

    body {

      margin: 0;

      font-family: Roboto;

      background-color: #f5f5f5;;

    }

  </style>

  <div id="app"></div>

</body>

</html>

import React, { memo } from 'react';

import { BrowserRouter } from 'react-router-dom';

import { Provider } from 'react-redux';

import { store } from 'store';

import { Router } from 'components/Router';

import { Profile } from 'components/Profile';

import { ErrorHandler } from 'components/ErrorHandler';

import { Header } from 'components/Header';

export const App = memo(() => (

  <>

    <Header />

    <Router />

  </>

));

export const AppContainer = memo(() => (

  <BrowserRouter>

    <Provider store={store}>

      <Profile>

        <ErrorHandler>

          <App />

        </ErrorHandler>

      </Profile>

    </Provider>

  </BrowserRouter>

));

import React, { useCallback, useState } from 'react';

import { connect } from 'react-redux';

import { withRouter } from 'react-router-dom';

import { flowRight } from 'lodash';

import { withStyles } from '@material-ui/core';

import Menu from '@material-ui/core/Menu';

import MenuItem from '@material-ui/core/MenuItem';

import Icon from '@material-ui/core/Icon';

import { isAllowed } from 'utils';

import { ROUTE, ROLE } from 'configs';

import { getFirstNameSelector, getLastNameSelector, getRoleSelector } from 'components/Profile';

import { MENU\_ID } from './constants';

const styles = () => ({

  header: {

    display: 'flex',

    justifyContent: 'space-between',

    backgroundColor: '#3b3a3c',

    height: '56px',

  },

  leftBar: {

    display: 'flex',

    alignItems: 'stretch',

  },

  rightBar: {

    display: 'flex',

    alignItems: 'stretch',

    color: 'white',

  },

  profile: {

    display: 'flex',

    alignItems: 'center',

    '& > :first-child': {

      marginRight: '10px',

    }

  },

  logo: {

    display: 'flex',

    alignItems: 'center',

    color: 'white',

    backgroundColor: '#282728',

    alignSelf: 'stretch',

    padding: '0 24px',

    cursor: 'pointer',

    fontSize: '17px',

    textTransform: 'uppercase',

    '& > :first-child': {

      fontSize: '32px',

      marginRight: '10px',

    },

  },

  headerButton: {

    display: 'flex',

    alignItems: 'center',

    padding: '0 24px',

    color: 'white',

    cursor: 'pointer',

    '&:hover': {

      backgroundColor: '#282728',

    }

  },

});

export const HeaderComponent = ({ history, classes = {}, firstName, lastName, role }) => {

  const [anchorEl, setAnchorEl] = useState(null);

  const [menuId, setMenuId] = useState(null);

  const handleClick = useCallback((event) => {

    setAnchorEl(event.currentTarget);

    setMenuId(Number(event.currentTarget.dataset.menuId));

  }, [setAnchorEl, setMenuId]);

  const handleClose = useCallback(() => {

    setAnchorEl(null);

    setMenuId(null);

  }, [setAnchorEl, setMenuId]);

  const handleMenuItemClick = useCallback((event) => {

    history.push(event.currentTarget.dataset.route);

    handleClose();

  }, [history, handleClose]);

  return (

    <div className={classes.header}>

      <div className={classes.leftBar}>

        <div className={classes.logo}>

          <Icon>healing</Icon>

          <div>Home treatment</div>

        </div>

        {isAllowed(role, [ROLE.USER, ROLE.ADMIN]) &&

          <div className={classes.headerButton} data-route={ROUTE.ILLNESS} onClick={handleMenuItemClick}>Search illnesses</div>

        }

        {isAllowed(role, [ROLE.ADMIN]) &&

          <div className={classes.headerButton} data-menu-id={MENU\_ID.SYMPTOM} onClick={handleClick}>Management</div>

        }

        <Menu

          anchorEl={anchorEl}

          keepMounted

          open={menuId === MENU\_ID.SYMPTOM}

          onClose={handleClose}

        >

          <MenuItem data-route={ROUTE.ILLNESS\_CREATE} onClick={handleMenuItemClick}>Create illness</MenuItem>

          <MenuItem data-route={ROUTE.ILLNESS\_MANAGE} onClick={handleMenuItemClick}>Manage illness</MenuItem>

          <MenuItem data-route={ROUTE.SYMPTOM\_MANAGE} onClick={handleMenuItemClick}>Manage symptoms</MenuItem>

        </Menu>

      </div>

      <div className={classes.rightBar}>

        {firstName && (

          <div className={`${classes.profile} ${classes.headerButton}`} data-menu-id={MENU\_ID.PROFILE} onClick={handleClick}>

            <div>{firstName} {lastName}</div>

            <Icon>account\_circle</Icon>

          </div>

        )}

        <Menu

          anchorEl={anchorEl}

          keepMounted

          open={menuId === MENU\_ID.PROFILE}

          onClose={handleClose}

        >

          <MenuItem>Profile</MenuItem>

          <MenuItem>Logout</MenuItem>

        </Menu>

      </div>

    </div>

  );

};

const mapStateToProps = state => ({

  firstName: getFirstNameSelector(state),

  lastName: getLastNameSelector(state),

  role: getRoleSelector(state),

});

const connector = connect(mapStateToProps);

export const Header = flowRight(

  connector,

  withRouter,

  withStyles(styles),

)(HeaderComponent);

import { ISymptom, IMedicine, IRecommendation } from 'models';

export const GET\_SYMPTOMS\_SUCCESS = '[ILLNESS\_CREATE]: get symptoms success';

export const GET\_MEDICINES\_SUCCESS = '[ILLNESS\_CREATE]: get medicines success';

export const GET\_RECOMMENDATIONS\_SUCCESS = '[ILLNESS\_CREATE]: get recommendations success';

export const CHANGE\_IS\_FETCHING = '[ILLNESS\_CREATE]: change isFetching';

export const CREATE\_ILLNESS\_REQUEST = '[ILLNESS\_CREATE]: create illness request';

export const CREATE\_ILLNESS\_SUCCESS = '[ILLNESS\_CREATE]: create illness success';

export const CREATE\_ILLNESS\_FAILURE = '[ILLNESS\_CREATE]: create illness failure';

export const getSymptomsSuccessAction = (symptomsList: ISymptom[]) => ({

  type: GET\_SYMPTOMS\_SUCCESS,

  payload: { symptomsList },

});

export const getMedicinesSuccessAction = (medicinesList: IMedicine[]) => ({

  type: GET\_MEDICINES\_SUCCESS,

  payload: { medicinesList },

});

export const getRecommendationsSuccessAction = (recommendationsList: IRecommendation[]) => ({

  type: GET\_RECOMMENDATIONS\_SUCCESS,

  payload: { recommendationsList },

});

export const changeIsFetchingAction = (isFetching: boolean) => ({

  type: CHANGE\_IS\_FETCHING,

  payload: { isFetching },

});

export const createIllnessRequestAction = () => ({

  type: CREATE\_ILLNESS\_REQUEST,

  payload: { isFetching: true },

});

export const createIllnessSuccessAction = () => ({

  type: CREATE\_ILLNESS\_SUCCESS,

  payload: { isFetching: false },

});

export const createIllnessFailureAction = () => ({

  type: CREATE\_ILLNESS\_FAILURE,

  payload: { isFetching: false },

});

export type Actions = ReturnType<

  | typeof getSymptomsSuccessAction

  | typeof getMedicinesSuccessAction

  | typeof getRecommendationsSuccessAction

  | typeof changeIsFetchingAction

  | typeof createIllnessRequestAction

  | typeof createIllnessSuccessAction

  | typeof createIllnessFailureAction

>;

import { ThunkAction } from 'redux-thunk';

import { IStore } from 'store';

import { ISymptom, IMedicine, IRecommendation, ICreateIllness } from 'models';

import { http } from 'utils';

import { PATH } from 'configs';

import {

  Actions,

  getSymptomsSuccessAction,

  getMedicinesSuccessAction,

  getRecommendationsSuccessAction,

  changeIsFetchingAction,

  createIllnessRequestAction,

  createIllnessSuccessAction,

  createIllnessFailureAction,

} from './actions';

type ThunkResult<T> = ThunkAction<T, IStore, undefined, Actions>;

export const getFilterDataAction = (): ThunkResult<void> => (dispatch) => {

  dispatch(changeIsFetchingAction(true));

  return Promise.all([

    http.get<ISymptom[]>(PATH.SYMPTOM\_SHORT),

    http.get<IMedicine[]>(PATH.MEDICINE),

    http.get<IRecommendation[]>(PATH.RECOMMENDATION)

  ])

    .then(([symptoms, medicines, recommendations]) => {

      dispatch(getSymptomsSuccessAction(symptoms));

      dispatch(getMedicinesSuccessAction(medicines));

      dispatch(getRecommendationsSuccessAction(recommendations));

      dispatch(changeIsFetchingAction(false));

    })

    .catch(() => dispatch(changeIsFetchingAction(false)));

};

export const createIllnessAction = (illness: ICreateIllness): ThunkResult<void> => (dispatch) => {

  dispatch(createIllnessRequestAction());

  return http.post(PATH.ILLNESS, illness)

    .then(() => dispatch(createIllnessSuccessAction()))

    .catch(() => dispatch(createIllnessFailureAction()));

};

import React, { memo } from 'react';

import { compose } from 'redux';

import { withStyles } from '@material-ui/core';

import { IllnessCreateForm } from 'components/common/forms';

import { Button } from 'components/common/Button';

import { Toolbar } from 'components/common/Toolbar';

import { Page } from 'components/common/Page';

const styles = () => ({

  createButton: {

    marginTop: '20px',

  }

});

export const IllnessCreateComponent = ({ symptoms, medicines, recommendations, onSubmitForm, onCreateIllness, classes }) => (

  <div>

    <Toolbar>

      <div>Create illness</div>

    </Toolbar>

    <Page>

      <IllnessCreateForm

        symptoms={symptoms}

        medicines={medicines}

        recommendations={recommendations}

        onSubmit={onCreateIllness}

      />

      <Button className={classes.createButton} onClick={onSubmitForm}>Create</Button>

    </Page>

  </div>

);

export const IllnessCreate = compose(

  withStyles(styles),

  memo,

)(IllnessCreateComponent);

import React, { useEffect } from 'react';

import { connect } from 'react-redux';

import { submit } from 'redux-form';

import { ICreateIllness } from 'models';

import { FORM\_NAME } from 'components/common/forms/IllnessCreateForm';

import { IllnessCreate } from './IllnessCreate';

import { getFilterDataAction, createIllnessAction } from './epicActions';

import { getNormalizedSymptomsSelector, getNormalizedMedicinesSelector, getNormalizedRecommendationsSelector } from './selectors';

export const IllnessCreateContainerComponent = ({ symptoms, medicines, recommendations, getFilterData, submitForm, createIllness }) => {

  useEffect(() => {

    getFilterData();

  }, [getFilterData]);

  return (

    <IllnessCreate

      symptoms={symptoms}

      medicines={medicines}

      recommendations={recommendations}

      onSubmitForm={submitForm}

      onCreateIllness={createIllness}

    />

  );

};

const mapStateToProps = state => ({

  symptoms: getNormalizedSymptomsSelector(state),

  medicines: getNormalizedMedicinesSelector(state),

  recommendations: getNormalizedRecommendationsSelector(state),

});

const mapDispatchToProps = (dispatch) => ({

  getFilterData: () => dispatch(getFilterDataAction()),

  submitForm: () => dispatch(submit(FORM\_NAME)),

  createIllness: (illness: ICreateIllness) => dispatch(createIllnessAction(illness)),

});

const connector = connect(mapStateToProps, mapDispatchToProps);

export const IllnessCreateContainer = connector(IllnessCreateContainerComponent);

import { ISymptom, IMedicine, IRecommendation } from 'models';

import {

  Actions,

  GET\_SYMPTOMS\_SUCCESS,

  GET\_MEDICINES\_SUCCESS,

  GET\_RECOMMENDATIONS\_SUCCESS,

  CHANGE\_IS\_FETCHING,

  CREATE\_ILLNESS\_REQUEST,

  CREATE\_ILLNESS\_SUCCESS,

  CREATE\_ILLNESS\_FAILURE,

} from './actions';

export interface IIllnessCreateState {

  isFetching: boolean;

  symptomsList: ISymptom[];

  medicinesList: IMedicine[];

  recommendationsList: IRecommendation[];

}

const initialState: IIllnessCreateState = {

  isFetching: false,

  symptomsList: [],

  medicinesList: [],

  recommendationsList: [],

};

export const illnessCreate = (state: IIllnessCreateState = initialState, { type, payload }: Actions): IIllnessCreateState => {

  switch(type) {

    case GET\_SYMPTOMS\_SUCCESS:

    case GET\_MEDICINES\_SUCCESS:

    case GET\_RECOMMENDATIONS\_SUCCESS:

    case CHANGE\_IS\_FETCHING:

    case CREATE\_ILLNESS\_REQUEST:

    case CREATE\_ILLNESS\_SUCCESS:

    case CREATE\_ILLNESS\_FAILURE:

      return {

        ...state,

        ...payload,

      };

    default:

      return state;

  }

};

import { createSelector } from 'reselect';

import { IStore } from 'store';

const getIllnessCreate = (state: IStore) => state.illness.illnessCreate;

export const getSymptomsListSelector = createSelector(

  getIllnessCreate,

  illnessCreate => illnessCreate.symptomsList,

);

export const getNormalizedSymptomsSelector = createSelector(

  getSymptomsListSelector,

  symptomsList => [{ id: '' }, ...symptomsList],

);

export const getMedicinesSelector = createSelector(

  getIllnessCreate,

  illnessCreate => illnessCreate.medicinesList,

);

export const getNormalizedMedicinesSelector = createSelector(

  getMedicinesSelector,

  medicinesList => [{ id: '' }, ...medicinesList],

);

export const getRecommendationsSelector = createSelector(

  getIllnessCreate,

  illnessCreate => illnessCreate.recommendationsList,

);

export const getNormalizedRecommendationsSelector = createSelector(

  getRecommendationsSelector,

  recommendationsList => [{ id: '' }, ...recommendationsList],

);

import { IIllness } from 'models';

export const GET\_ILLNESS\_DETAILS\_REQUEST = '[ILLNESS\_DETAILS]: get illness details request';

export const GET\_ILLNESS\_DETAILS\_SUCCESS = '[ILLNESS\_DETAILS]: get illness details success';

export const GET\_ILLNESS\_DETAILS\_FAILURE = '[ILLNESS\_DETAILS]: get illness details failure';

export const getIllnessDetailsRequestAction = () => ({

  type: GET\_ILLNESS\_DETAILS\_REQUEST,

  payload: { isFetching: true },

});

export const getIllnessDetailsSuccessAction = (illnessInfo: IIllness) => ({

  type: GET\_ILLNESS\_DETAILS\_SUCCESS,

  payload: { isFetching: false, illnessInfo },

});

export const getIllnessDetailsFailureAction = () => ({

  type: GET\_ILLNESS\_DETAILS\_FAILURE,

  payload: { isFetching: false },

});

export type Actions = ReturnType<

  typeof getIllnessDetailsRequestAction

  | typeof getIllnessDetailsSuccessAction

  | typeof getIllnessDetailsFailureAction

>;

import { ThunkAction } from 'redux-thunk';

import { IStore } from 'store';

import { http } from 'utils';

import { PATH } from 'configs';

import { IIllness } from 'models';

import {

  Actions,

  getIllnessDetailsRequestAction,

  getIllnessDetailsSuccessAction,

  getIllnessDetailsFailureAction,

} from './actions';

type ThunkResult<T> = ThunkAction<T, IStore, undefined, Actions>;

export const getIllnessDetailsAction = (id: number): ThunkResult<void> => (dispatch) => {

  dispatch(getIllnessDetailsRequestAction());

  return http.get<IIllness>(`${PATH.ILLNESS}/${id}`)

    .then(data => dispatch(getIllnessDetailsSuccessAction(data)))

    .catch(() => dispatch(getIllnessDetailsFailureAction()));

};

import React, { memo, useMemo } from 'react';

import { Toolbar } from 'components/common/Toolbar';

import { Page } from 'components/common/Page';

import { TextEditor } from 'components/common/TextEditor';

import { Button } from 'components/common/Button';

import { IMedicine, IRecommendation, ISymptom } from 'models';

interface IProps {

  name: string;

  description: string;

  updatedAt: string;

  medicines: IMedicine[];

  recommendations: IRecommendation[];

  symptoms: ISymptom[];

}

export const IllnessDetailsComponent = ({ name, description, symptoms = [], recommendations = [], medicines = [], updatedAt, onLeaveFeedback }: IProps) => {

  const textEditorConfig = useMemo(() => ({ readonly: true, toolbar: false, showXPathInStatusbar: false, showWordsCounter: false, showCharsCounter: false }), []);

  return (

    <div>

      <Toolbar>

        <div>Illness details</div>

        <div>{name}</div>

      </Toolbar>

      <Page>

        <TextEditor value={description} config={textEditorConfig} />

        <h3>Symptoms</h3>

        <ul>

          {symptoms.map(({ id, name }) => <li key={id}>{name}</li>)}

        </ul>

        <h3>Recommendations</h3>

        <ul>

          {recommendations.map(({ id, name }) => <li key={id}>{name}</li>)}

        </ul>

        <h3>Medicines</h3>

        <ul>

          {medicines.map(({ id, name }) => <li key={id}>{name}</li>)}

        </ul>

        <div>{updatedAt}</div>

        <Button color="secondary" onClick={onLeaveFeedback}>Leave feedback</Button>

      </Page>

    </div>

  );

};

export const IllnessDetails = memo(IllnessDetailsComponent);

import React, { useEffect, useCallback } from 'react';

import { connect } from 'react-redux';

import { IIllness } from 'models';

import { ROUTE } from 'configs';

import { IllnessDetails } from './IllnessDetails';

import { getIllnessDetailsAction } from './epicActions';

import { getIsFetchingSelector, getIllnessInfoSelector } from './selectors';

interface IProps {

  match: any;

  illnessInfo: IIllness;

  getIllnessDetails: any;

}

export const IllnessDetailsContainerComponent = ({ match: { params: { id } }, history, illnessInfo, getIllnessDetails }: IProps) => {

  useEffect(() => {

    getIllnessDetails(id);

  }, []);

  const handleLeaveFeedback = useCallback(() => history.push(`${ROUTE.ILLNESS\_REVIEW\_CREATE}/${id}`), [history, id]);

  return (

    <IllnessDetails

      name={illnessInfo.name}

      description={illnessInfo.description}

      symptoms={illnessInfo.symptoms}

      medicines={illnessInfo.medicines}

      recommendations={illnessInfo.recommendations}

      updatedAt={illnessInfo.updatedAt}

      onLeaveFeedback={handleLeaveFeedback}

    />

  );

};

const mapStateToProps = state => ({

  isFetching: getIsFetchingSelector(state),

  illnessInfo: getIllnessInfoSelector(state),

});

const mapDispatchToProps = {

  getIllnessDetails: getIllnessDetailsAction,

};

const connector = connect(mapStateToProps, mapDispatchToProps);

export const IllnessDetailsContainer = connector(IllnessDetailsContainerComponent);

import { IIllness, ISymptom, IMedicine, IRecommendation } from 'models';

export const GET\_ILLNESS\_SUCCESS = '[ILLNESS\_EDIT]: get illness success';

export const GET\_SYMPTOMS\_SUCCESS = '[ILLNESS\_EDIT]: get symptoms success';

export const GET\_MEDICINES\_SUCCESS = '[ILLNESS\_EDIT]: get medicines success';

export const GET\_RECOMMENDATIONS\_SUCCESS = '[ILLNESS\_EDIT]: get recommendations success';

export const CHANGE\_IS\_FETCHING = '[ILLNESS\_EDIT]: change isFetching';

export const getIllnessSuccessAction = (illnessInfo: IIllness) => ({

  type: GET\_ILLNESS\_SUCCESS,

  payload: { illnessInfo },

});

export const getSymptomsSuccessAction = (symptomsList: ISymptom[]) => ({

  type: GET\_SYMPTOMS\_SUCCESS,

  payload: { symptomsList },

});

export const getMedicinesSuccessAction = (medicinesList: IMedicine[]) => ({

  type: GET\_MEDICINES\_SUCCESS,

  payload: { medicinesList },

});

export const getRecommendationsSuccessAction = (recommendationsList: IRecommendation[]) => ({

  type: GET\_RECOMMENDATIONS\_SUCCESS,

  payload: { recommendationsList },

});

export const changeIsFetchingAction = (isFetching: boolean) => ({

  type: CHANGE\_IS\_FETCHING,

  payload: { isFetching },

});

export type Actions = ReturnType<

  typeof getIllnessSuccessAction

  | typeof getSymptomsSuccessAction

  | typeof getMedicinesSuccessAction

  | typeof getRecommendationsSuccessAction

  | typeof changeIsFetchingAction

>;

import { ThunkAction } from 'redux-thunk';

import { IStore } from 'store';

import { http } from 'utils';

import { PATH } from 'configs';

import { IIllness, IMedicine, IRecommendation, ISymptom } from 'models';

import {

  Actions,

  getIllnessSuccessAction,

  getSymptomsSuccessAction,

  getMedicinesSuccessAction,

  getRecommendationsSuccessAction,

  changeIsFetchingAction,

} from './actions';

type ThunkResult<T> = ThunkAction<T, IStore, undefined, Actions>;

export const getInitialDataAction = (id: number): ThunkResult<void> => (dispatch) => {

  dispatch(changeIsFetchingAction(true));

  Promise.all([

    http.get<IIllness>(`${PATH.ILLNESS}/${id}`),

    http.get<ISymptom[]>(PATH.SYMPTOM\_SHORT),

    http.get<IMedicine[]>(PATH.MEDICINE),

    http.get<IRecommendation[]>(PATH.RECOMMENDATION),

  ])

    .then(([illness, symptoms, medicines, recommendations]) => {

      dispatch(getIllnessSuccessAction(illness));

      dispatch(getSymptomsSuccessAction(symptoms));

      dispatch(getMedicinesSuccessAction(medicines));

      dispatch(getRecommendationsSuccessAction(recommendations));

      dispatch(changeIsFetchingAction(false));

    })

    .catch(() => dispatch(changeIsFetchingAction(false)));

};

import React, { memo } from 'react';

import { IllnessCreateForm } from 'components/common/forms';

import { Toolbar } from 'components/common/Toolbar';

import { Button } from 'components/common/Button';

import { Page } from 'components/common/Page';

export const IllnessEditComponent = ({ illness, symptoms, medicines, recommendations }) => (

  <div>

    <Toolbar>

      <div>Edit illness</div>

    </Toolbar>

    <Page>

      <IllnessCreateForm

        initialValues={illness}

        symptoms={symptoms}

        medicines={medicines}

        recommendations={recommendations}

      />

      <Button>Save</Button>

    </Page>

  </div>

);

export const IllnessEdit = memo(IllnessEditComponent);

import React, { useEffect } from 'react';

import { connect } from 'react-redux';

import { IllnessEdit } from './IllnessEdit';

import { getInitialDataAction } from './epicActions';

import {

  getNormalizedIllnessSelector,

  getSymptomsListSelector,

  getMedicinesSelector,

  getRecommendationsSelector,

} from './selectors';

export const IllnessEditContainerComponent = ({ illness, symptoms, medicines, recommendations, getInitialData }) => {

  useEffect(() => {

    getInitialData();

  }, []);

  return <IllnessEdit illness={illness} symptoms={symptoms} medicines={medicines} recommendations={recommendations} />

};

const mapStateToProps = state => ({

  illness: getNormalizedIllnessSelector(state),

  symptoms: getSymptomsListSelector(state),

  medicines: getMedicinesSelector(state),

  recommendations: getRecommendationsSelector(state),

});

const mapDispatchToProps = (dispatch, { match: { params: { id } } }) => ({

  getInitialData: () => dispatch(getInitialDataAction(id)),

});

const connector = connect(mapStateToProps, mapDispatchToProps);

export const IllnessEditContainer = connector(IllnessEditContainerComponent);

import { ThunkAction } from 'redux-thunk';

import { IStore } from 'store';

import {IIllnessShort, ISymptom} from 'models';

import { http } from 'utils';

import { PATH } from 'configs';

import {

  Actions,

  getSymptomsRequestAction,

  getSymptomsSuccessAction,

  getSymptomsFailureAction,

  getIllnessesRequestAction,

  getIllnessesSuccessAction,

  getIllnessesFailureAction,

} from './actions';

type ThunkResult<T> = ThunkAction<T, IStore, undefined, Actions>;

export const getSymptomsAction = (): ThunkResult<void> => (dispatch) => {

  dispatch(getSymptomsRequestAction());

  return http.get<ISymptom[]>(PATH.SYMPTOM\_SHORT)

    .then(data => dispatch(getSymptomsSuccessAction(data)))

    .catch(() => dispatch(getSymptomsFailureAction()));

};

export const getIllnessesBySymptomIdsAction = (ids: number[]): ThunkResult<void> => (dispatch) => {

  dispatch(getIllnessesRequestAction());

  const params = ids.map(id => `ids=${id}`).join('&');

  return http.get<IIllnessShort[]>(`${PATH.ILLNESS\_SYMPTOMS}?${params}`)

    .then(data => dispatch(getIllnessesSuccessAction(data)))

    .catch(() => dispatch(getIllnessesFailureAction()));

};

import React, { useEffect, useCallback } from 'react';

import { connect } from 'react-redux';

import { IllnessList } from './lllnessList';

import { getSymptomsAction, getIllnessesBySymptomIdsAction } from './epicActions';

import { getSymptomsListSelector, getIllnessesListSelector } from './selectors';

export const IllnessListContainerComponent = ({ symptomsList, illnessesList, getSymptoms, getIllnessesBySymptomIds }) => {

  useEffect(() => {

    getSymptoms();

  }, []);

  return (

    <IllnessList

      symptoms={symptomsList}

      illnesses={illnessesList}

      onIllnessesFilter={getIllnessesBySymptomIds}

    />

  );

};

const mapStateToProps = state => ({

  symptomsList: getSymptomsListSelector(state),

  illnessesList: getIllnessesListSelector(state),

});

const mapDispatchToProps = {

  getSymptoms: getSymptomsAction,

  getIllnessesBySymptomIds: getIllnessesBySymptomIdsAction,

};

const connector = connect(mapStateToProps, mapDispatchToProps);

export const IllnessListContainer = connector(IllnessListContainerComponent);

import React, { useCallback } from 'react';

import { Toolbar } from 'components/common/Toolbar';

import { Page } from 'components/common/Page';

import { IllnessListFilter } from './IllnessListFilter';

import { IllnessListTable } from './IllnessListTable';

export const IllnessList = ({ symptoms, illnesses, onIllnessesFilter }) => {

  const handleIllnessesFilterSubmit = useCallback(({ symptoms }) => onIllnessesFilter(symptoms), [onIllnessesFilter]);

  return (

    <div>

      <Toolbar>

        <div>Search illness</div>

      </Toolbar>

      <Page>

        <IllnessListFilter symptoms={symptoms} onSubmit={handleIllnessesFilterSubmit} />

      </Page>

      <Page padding={false} marginTop={false}>

        <IllnessListTable illnesses={illnesses} />

      </Page>

    </div>

  );

};

import { ISymptom, IIllnessShort } from 'models';

import {

  Actions,

  GET\_SYMPTOMS\_REQUEST,

  GET\_SYMPTOMS\_SUCCESS,

  GET\_SYMPTOMS\_FAILURE,

  GET\_ILLNESSES\_REQUEST,

  GET\_ILLNESSES\_SUCCESS,

  GET\_ILLNESSES\_FAILURE,

} from './actions';

export interface IIllnessListState {

  isFetching: boolean;

  symptomsList: ISymptom[];

  illnessesList: IIllnessShort[];

}

const initialState: IIllnessListState = {

  isFetching: false,

  symptomsList: [],

  illnessesList: [],

};

export const illnessList = (state: IIllnessListState = initialState, { type, payload }: Actions): IIllnessListState => {

  switch (type) {

    case GET\_SYMPTOMS\_REQUEST:

    case GET\_SYMPTOMS\_SUCCESS:

    case GET\_SYMPTOMS\_FAILURE:

    case GET\_ILLNESSES\_REQUEST:

    case GET\_ILLNESSES\_SUCCESS:

    case GET\_ILLNESSES\_FAILURE:

      return {

        ...state,

        ...payload,

      };

    default:

      return state;

  }

};

import { createSelector } from 'reselect';

import { IStore } from 'store';

const getIllnessList = (state: IStore) => state.illness.illnessList;

export const getSymptomsListSelector = createSelector(

  getIllnessList,

  illnessList => illnessList.symptomsList,

);

export const getIllnessesListSelector = createSelector(

  getIllnessList,

  illnessList => illnessList.illnessesList,

);

import React, { useCallback } from 'react';

import { reduxForm } from 'redux-form';

import { withStyles } from '@material-ui/core';

import { RFields } from 'components/common/RFields';

import { Button } from 'components/common/Button';

import { validate } from './validators';

import { FORM\_NAME, FORM\_FIELDS } from './constants';

const styles = () => ({

  filters: {

    display: 'flex',

    flexWrap: 'wrap',

    '& > div': {

      width: '33%',

    }

  },

  submitButton: {

    marginRight: '10px',

  },

  actions: {

    display: 'flex',

    justifyContent: 'space-between',

    alignItems: 'center',

  },

  error: {

    color: '#f44336',

  }

});

export const IllnessListFilterComponent = ({ handleSubmit, array, symptoms, error, submitFailed, classes }) => {

  const handleAddSymptom = useCallback(() => array.push(FORM\_FIELDS.SYMPTOMS, ''), [array]);

  return (

    <form onSubmit={handleSubmit}>

      <RFields.ArraySelect

        name={FORM\_FIELDS.SYMPTOMS}

        data={symptoms}

        className={classes.filters}

        label="Symptom"

      />

      <div className={classes.actions}>

        <div>

          <Button className={classes.submitButton} type="submit">Search</Button>

          <Button color="secondary" type="button" onClick={handleAddSymptom}>Add symptom</Button>

        </div>

        <div className={classes.error}>{submitFailed && error}</div>

      </div>

    </form>

  );

};

const formCreator = reduxForm({

  form: FORM\_NAME,

  validate,

  initialValues: {

    [FORM\_FIELDS.SYMPTOMS]: [''],

  },

});

export const IllnessListFilter = withStyles(styles)(formCreator(IllnessListFilterComponent));

export const validate = ({ symptoms }) => {

  if (symptoms && !symptoms.find(id => id !== '')) {

    return { \_error: 'Please select at least one symptom' };

  }

  return {};

};

import React, { memo } from 'react';

import Paper from "@material-ui/core/Paper/Paper";

import Table from "@material-ui/core/Table";

import TableHead from "@material-ui/core/TableHead";

import TableRow from "@material-ui/core/TableRow";

import TableCell from "@material-ui/core/TableCell/TableCell";

import TableBody from "@material-ui/core/TableBody";

import TableContainer from "@material-ui/core/TableContainer";

import Link from "@material-ui/core/Link";

import { IIllnessShort } from 'models';

import { ROUTE } from 'configs';

import { Rating } from 'components/common/Rating';

interface IProps {

  illnesses: IIllnessShort[],

}

export const IllnessListTableComponent = ({ illnesses }: IProps) => (

  <TableContainer component={Paper}>

    <Table>

      {Boolean(illnesses.length) &&

        <TableHead>

          <TableRow>

            <TableCell align="left">Name</TableCell>

            <TableCell align="right">Rating</TableCell>

          </TableRow>

        </TableHead>

      }

      <TableBody>

        {illnesses.map(({ id, name, rating }) => (

          <TableRow key={id}>

            <TableCell align="left">

              <Link href={`${ROUTE.ILLNESS\_DETAILS}/${id}`} target="\_blank">{name}</Link>

            </TableCell>

            <TableCell align="right">

              <Rating value={rating} readOnly />

            </TableCell>

          </TableRow>

        ))}

      </TableBody>

    </Table>

  </TableContainer>

);

export const IllnessListTable = memo(IllnessListTableComponent);

import { IIllnessShort } from 'models';

export const GET\_ILLNESSES\_REQUEST = '[ILLNESS\_MANAGE]: get illnesses request';

export const GET\_ILLNESSES\_SUCCESS = '[ILLNESS\_MANAGE]: get illnesses success';

export const GET\_ILLNESSES\_FAILURE = '[ILLNESS\_MANAGE]: get illnesses failure';

export const DELETE\_ILLNESS\_REQUEST = '[ILLNESS\_MANAGE]: delete illness request';

export const DELETE\_ILLNESS\_SUCCESS = '[ILLNESS\_MANAGE]: delete illness success';

export const DELETE\_ILLNESS\_FAILURE = '[ILLNESS\_MANAGE]: delete illness failure';

export const getIllnessesRequestAction = () => ({

  type: GET\_ILLNESSES\_REQUEST,

  payload: { isFetching: true },

});

export const getIllnessesSuccessAction = (illnessesList: IIllnessShort[]) => ({

  type: GET\_ILLNESSES\_SUCCESS,

  payload: { isFetching: false, illnessesList },

});

export const getIllnessesFailureAction = () => ({

  type: GET\_ILLNESSES\_FAILURE,

  payload: { isFetching: false },

});

export const deleteIllnessRequestAction = () => ({

  type: DELETE\_ILLNESS\_REQUEST,

  payload: { isFetching: true },

});

export const deleteIllnessSuccessAction = (id: number) => ({

  type: DELETE\_ILLNESS\_SUCCESS,

  payload: { isFetching: false, id },

});

export const deleteIllnessFailureAction = () => ({

  type: DELETE\_ILLNESS\_FAILURE,

  payload: { isFetching: false },

});

export type Actions = ReturnType<

  typeof getIllnessesRequestAction

  | typeof getIllnessesSuccessAction

  | typeof getIllnessesSuccessAction

  | typeof deleteIllnessRequestAction

  | typeof deleteIllnessSuccessAction

  | typeof deleteIllnessFailureAction

>;

import { ThunkAction } from 'redux-thunk';

import { IStore } from 'store';

import { IIllnessShort }  from 'models';

import { http } from 'utils';

import { PATH } from 'configs';

import {

  Actions,

  getIllnessesRequestAction,

  getIllnessesSuccessAction,

  getIllnessesFailureAction,

  deleteIllnessRequestAction,

  deleteIllnessSuccessAction,

  deleteIllnessFailureAction,

} from './actions';

type ThunkResult<T> = ThunkAction<T, IStore, undefined, Actions>;

export const getIllnessesAction = (): ThunkResult<void> => (dispatch) => {

  dispatch(getIllnessesRequestAction());

  http.get<IIllnessShort[]>(PATH.ILLNESS\_SHORT)

    .then(data => dispatch(getIllnessesSuccessAction(data)))

    .catch(() => dispatch(getIllnessesFailureAction()));

};

export const deleteIllnessAction = (id: number): ThunkResult<void> => (dispatch) => {

  dispatch(deleteIllnessRequestAction());

  http.delete<void>(`${PATH.ILLNESS}/${id}`)

    .then(() => dispatch(deleteIllnessSuccessAction(id)))

    .catch(() => dispatch(deleteIllnessFailureAction()));

};

import React, { memo } from 'react';

import { Toolbar } from 'components/common/Toolbar';

import { Page } from 'components/common/Page';

import { Button } from 'components/common/Button';

import { IllnessManageTable } from './IllnessManageTable';

export const IllnessManageComponent = ({ illnesses, onCreate, onEdit, onDelete, onReview }) => (

  <div>

    <Toolbar>

      <div>Illness manage</div>

      <Button onClick={onCreate}>Create</Button>

    </Toolbar>

    <Page padding={false}>

      <IllnessManageTable illnesses={illnesses} onEdit={onEdit} onDelete={onDelete} onReview={onReview} />

    </Page>

  </div>

);

export const IllnessManage = memo(IllnessManageComponent);

import React, { useEffect, useCallback } from 'react';

import { connect } from 'react-redux';

import { ROUTE } from 'configs';

import { IllnessManage } from './IllnessManage';

import { getIllnessesAction, deleteIllnessAction } from './epicActions';

import { getNormalizedIllnessesSelector } from './selectors';

export const IllnessManageContainerComponent = ({ illnesses, history, getIllnesses, deleteIllness }) => {

  useEffect(() => {

    getIllnesses();

  }, []);

  const handleEdit = useCallback(id => history.push(`${ROUTE.ILLNESS\_EDIT}/${id}`), [history]);

  const handleCreate = useCallback(() => history.push(ROUTE.ILLNESS\_CREATE), [history]);

  const handleReview = useCallback(id => history.push(`${ROUTE.ILLNESS\_REVIEW}/${id}`), [history]);

  return <IllnessManage illnesses={illnesses} onCreate={handleCreate} onEdit={handleEdit} onDelete={deleteIllness} onReview={handleReview} />

};

const mapStateToProps = state => ({

  illnesses: getNormalizedIllnessesSelector(state),

});

const mapDispatchToProps = ({

  getIllnesses: getIllnessesAction,

  deleteIllness: deleteIllnessAction,

});

const connector = connect(mapStateToProps, mapDispatchToProps);

export const IllnessManageContainer = connector(IllnessManageContainerComponent);

import React, { memo, useCallback } from 'react';

import { flowRight } from 'lodash';

import { withStyles } from '@material-ui/core';

import Table from '@material-ui/core/Table';

import TableBody from '@material-ui/core/TableBody';

import TableCell from '@material-ui/core/TableCell';

import TableContainer from '@material-ui/core/TableContainer';

import TableHead from '@material-ui/core/TableHead';

import TableRow from '@material-ui/core/TableRow';

import Paper from '@material-ui/core/Paper';

import Icon from '@material-ui/core/Icon';

const ACTION = {

  EDIT: 1,

  DELETE: 2,

  REVIEW: 3,

};

const styles = () => ({

  actionsIcon: {

    cursor: 'pointer',

    '&:hover': {

      color: '#0000009e',

    },

  },

  editIcon: {

    marginRight: '10px',

  }

});

export const IllnessManageTableComponent = ({ illnesses, classes, onEdit, onDelete, onReview }) => {

  const handleEditClick = useCallback(({ target: { dataset: { id, action } } }) => {

    switch(Number(action)) {

      case ACTION.EDIT:

        return onEdit(Number(id));

      case ACTION.DELETE:

        return onDelete(Number(id));

      case ACTION.REVIEW:

        return onReview(Number(id));

    }

  }, [onEdit, onDelete, onReview]);

  return (

    <TableContainer component={Paper} onClick={handleEditClick}>

      <Table>

        <TableHead>

          <TableRow>

            <TableCell align="left">Id</TableCell>

            <TableCell align="left">Name</TableCell>

            <TableCell align="right">Created at</TableCell>

            <TableCell align="right">Updated at</TableCell>

            <TableCell align="right">Actions</TableCell>

          </TableRow>

        </TableHead>

        <TableBody>

          {illnesses.map(({ id, name, createdAt, updatedAt }) => (

            <TableRow key={id}>

              <TableCell align="left">{id}</TableCell>

              <TableCell align="left">{name}</TableCell>

              <TableCell align="right">{createdAt}</TableCell>

              <TableCell align="right">{updatedAt}</TableCell>

              <TableCell align="right">

                <Icon className={`${classes.actionsIcon} ${classes.editIcon}`} data-id={id} data-action={ACTION.REVIEW}>analytics</Icon>

                <Icon className={`${classes.actionsIcon} ${classes.editIcon}`} data-id={id} data-action={ACTION.EDIT}>edit</Icon>

                <Icon className={classes.actionsIcon} data-id={id} data-action={ACTION.DELETE}>delete</Icon>

              </TableCell>

            </TableRow>

          ))}

        </TableBody>

      </Table>

    </TableContainer>

  );

};

export const IllnessManageTable = flowRight(

  withStyles(styles),

  memo,

)(IllnessManageTableComponent);

import React, { memo, useMemo } from 'react';

import { flowRight } from 'lodash';

import { LineChart, Line, Brush, XAxis, YAxis, CartesianGrid, Tooltip, ResponsiveContainer } from 'recharts';

import ExpansionPanel from '@material-ui/core/ExpansionPanel';

import ExpansionPanelSummary from '@material-ui/core/ExpansionPanelSummary';

import ExpansionPanelDetails from '@material-ui/core/ExpansionPanelDetails';

import { withStyles } from '@material-ui/core';

import { Toolbar } from 'components/common/Toolbar';

import { Page } from 'components/common/Page';

import { Rating } from 'components/common/Rating';

import { Select } from 'components/common/Select';

const styles = () => ({

  graphicsContainer: {

    padding: '10px 30px 20px 0',

  },

  medicines: {

    marginLeft: '65px',

    width: '200px',

  },

  expandedSummary: {

    width: '100%',

    display: 'flex',

    justifyContent: 'space-between',

    alignItems: 'center',

  },

  expandedDetails: {

    width: '100%',

    display: 'flex',

    justifyContent: 'space-between',

  },

  rating: {

    display: 'flex',

    justifyContent: 'space-between',

    alignItems: 'center',

    '& > div:first-child': {

      marginRight: '10px',

    }

  }

});

export const IllnessReviewComponent = ({ illnessName, medicines, illnessGraphicReviews, illnessReviews, illnessGraphicMedicines, onMedicineChange, classes }) => {

  const renderIllnessReviews = useMemo(() => illnessReviews.map(({ id, review, createdAt, descriptionRating, recommendationsRating, symptomsRating, medicines }) => (

    <ExpansionPanel key={id}>

      <ExpansionPanelSummary>

        <div className={classes.expandedSummary}>

          <div>{review}</div>

          <div>{createdAt}</div>

        </div>

      </ExpansionPanelSummary>

      <ExpansionPanelDetails>

        <div className={classes.expandedDetails}>

          <div>

            <div className={classes.rating}>

              <div>Description</div>

              <Rating value={descriptionRating} readOnly />

            </div>

            <div className={classes.rating}>

              <div>Recommendations</div>

              <Rating value={recommendationsRating} readOnly />

            </div>

            <div className={classes.rating}>

              <div>Symptoms</div>

              <Rating value={symptomsRating} readOnly />

            </div>

          </div>

          <div>

            {medicines.map(({ id, rating, medicine: { name } }) => (

              <div key={id} className={classes.rating}>

                <div>{name}</div>

                <Rating value={rating} readOnly />

              </div>

            ))}

          </div>

        </div>

      </ExpansionPanelDetails>

    </ExpansionPanel>

  )), [illnessReviews]);

  return (

    <div>

      <Toolbar>

        <div>Illness review</div>

        <div>{illnessName}</div>

      </Toolbar>

      <Page>

        <ResponsiveContainer width="100%" height={250}>

          <LineChart data={illnessGraphicReviews}>

            <CartesianGrid />

            <XAxis dataKey="date"/>

            <YAxis/>

            <Tooltip/>

            <Line type='monotone' dataKey="rating" stroke='#82ca9d' fill='#82ca9d' />

            <Brush />

          </LineChart>

        </ResponsiveContainer>

        <Select className={classes.medicines} data={medicines} onChange={onMedicineChange} />

        <ResponsiveContainer width="100%" height={250}>

          <LineChart data={illnessGraphicMedicines}>

            <CartesianGrid />

            <XAxis dataKey="date"/>

            <YAxis/>

            <Tooltip/>

            <Line type='monotone' dataKey="rating" stroke='#82ca9d' fill='#82ca9d' />

            <Brush />

          </LineChart>

        </ResponsiveContainer>

      </Page>

      <Page padding={false}>

        {renderIllnessReviews}

      </Page>

    </div>

  );

};

export const IllnessReview = flowRight(

  withStyles(styles),

  memo,

)(IllnessReviewComponent);

import React, { useEffect } from 'react';

import { connect } from 'react-redux';

import { flowRight } from 'lodash';

import { IllnessReview } from './IllnessReview';

import { changeMedicineIdFilterAction } from './actions';

import {

  getIllnessNameSelector,

  getIllnessGraphicReviewsSelector,

  getFilteredIllnessGraphicMedicinesSelector,

  getMedicinesSelector,

  getIllnessReviewsSelector,

} from './selectors';

import { getInitialDataAction } from './epicActions';

export const IllnessReviewContainerComponent = ({ illnessName, illnessGraphicReviews, illnessReviews, medicines, medicineIdFilter, illnessGraphicMedicines, getInitialData, changeMedicineIdFilter }) => {

  useEffect(() => {

    getInitialData();

  }, []);

  return <IllnessReview

    illnessName={illnessName}

    illnessGraphicReviews={illnessGraphicReviews}

    medicines={medicines}

    illnessReviews={illnessReviews}

    illnessGraphicMedicines={illnessGraphicMedicines}

    onMedicineChange={changeMedicineIdFilter}

  />;

};

const mapStateToProps = state => ({

  illnessName: getIllnessNameSelector(state),

  illnessGraphicReviews: getIllnessGraphicReviewsSelector(state),

  illnessGraphicMedicines: getFilteredIllnessGraphicMedicinesSelector(state),

  illnessReviews: getIllnessReviewsSelector(state),

  medicines: getMedicinesSelector(state),

});

const mapDispatchToProps = (dispatch, { match: { params: { id } } }) => ({

  getInitialData: () => dispatch(getInitialDataAction(id)),

  changeMedicineIdFilter: ({ target: { value } }) => dispatch(changeMedicineIdFilterAction(value)),

});

const connector = connect(mapStateToProps, mapDispatchToProps);

export const IllnessReviewContainer = flowRight(

  connector,

)(IllnessReviewContainerComponent);

import { ThunkAction } from 'redux-thunk';

import { reset } from 'redux-form';

import { FORM\_NAME } from 'components/common/forms/IllnessReviewCreateForm';

import { IStore } from 'store';

import { ICreateIllnessReview, IMedicine} from 'models';

import { http } from 'utils';

import { PATH, ROUTE } from 'configs';

import {

  Actions,

  getMedicinesRequestAction,

  getMedicinesSuccessAction,

  getMedicinesFailureAction,

  createReviewRequestAction,

  createReviewSuccessAction,

  createReviewFailureAction,

} from './actions';

type ThunkResult<T> = ThunkAction<T, IStore, undefined, Actions>;

export const getMedicinesAction = (illnessId: number): ThunkResult<void> => (dispatch) => {

  dispatch(getMedicinesRequestAction());

  http.get<IMedicine[]>(`${PATH.MEDICINE\_ILLNESS}/${illnessId}`)

    .then(data => dispatch(getMedicinesSuccessAction(data)))

    .catch(() => dispatch(getMedicinesFailureAction()));

};

export const createReviewAction = (illnessId: number, review: ICreateIllnessReview, history): ThunkResult<void> => (dispatch) => {

  dispatch(createReviewRequestAction());

  http.post<void>(PATH.ILLNESS\_REVIEW, { ...review, illnessId })

    .then(() => {

      dispatch(createReviewSuccessAction());

      dispatch(reset(FORM\_NAME));

      history.push(`${ROUTE.ILLNESS\_DETAILS}/${illnessId}`);

    })

    .catch(() => dispatch(createReviewFailureAction()));

};

import React, { memo } from 'react';

import { flowRight } from 'lodash';

import { withStyles } from '@material-ui/core';

import { Toolbar } from 'components/common/Toolbar';

import { IllnessReviewCreateForm } from 'components/common/forms';

const styles = () => ({

  review: {

    margin: '20px',

  },

});

export const IllnessReviewCreateComponent = ({ illnessReview, medicines, classes, onReviewCreate }) => (

  <div>

    <Toolbar>

      <div>Illness review</div>

    </Toolbar>

    <div className={classes.review}>

      <IllnessReviewCreateForm initialValues={illnessReview} medicines={medicines} onSubmit={onReviewCreate} />

    </div>

  </div>

);

export const IllnessReviewCreate = flowRight(

  withStyles(styles),

  memo,

)(IllnessReviewCreateComponent);

import React, { useEffect } from 'react';

import { connect } from 'react-redux';

import { submit } from 'redux-form';

import { FORM\_NAME } from 'components/common/forms/IllnessReviewCreateForm';

import { IllnessReviewCreate } from './IllnessReviewCreate';

import { createReviewAction, getMedicinesAction } from './epicActions';

import { getInitialIllnessReviewSelector, getMedicinesSelector } from './selectors';

export const IllnessReviewCreateContainerComponent = ({ medicines, initialIllnessReview, getMedicines, createReview, submitForm }) => {

  useEffect(() => {

    getMedicines();

  }, []);

  return(

    <IllnessReviewCreate

      medicines={medicines}

      illnessReview={initialIllnessReview}

      onReviewFormSubmit={submitForm}

      onReviewCreate={createReview}

    />

  );

};

const mapStateToProps = state => ({

  medicines: getMedicinesSelector(state),

  initialIllnessReview: getInitialIllnessReviewSelector(state),

});

const mapDispatchToProps = (dispatch, { history, match: { params: { id } } }) => ({

  submitForm: () => dispatch(submit(FORM\_NAME)),

  createReview: review => dispatch(createReviewAction(Number(id), review, history)),

  getMedicines: () => dispatch(getMedicinesAction(id)),

});

const connector = connect(mapStateToProps, mapDispatchToProps);

export const IllnessReviewCreateContainer = connector(IllnessReviewCreateContainerComponent);

import React, { useCallback, useState } from 'react';

import { reduxForm } from 'redux-form';

import { compose } from 'redux';

import { withStyles } from '@material-ui/core';

import { Page } from 'components/common/Page';

import { RFields } from 'components/common/RFields';

import { Button } from 'components/common/Button';

import { http, requiredValidator, emailValidator } from 'utils';

import { PATH, ROUTE } from 'configs';

import { FORM\_NAME, FORM\_FIELDS } from './constants';

export const LoginComponent = ({ handleSubmit, history, classes }) => {

  const [error, setError] = useState('');

  const handleSubmitForm = useCallback((value) => {

     http.post(PATH.LOGIN, value)

       .then(() => history.push(ROUTE.ILLNESS))

       .catch(() => setError('Wrong email or password'));

  }, []);

  return (

    <Page className={classes.form}>

      <form onSubmit={handleSubmit(handleSubmitForm)}>

        <RFields.Input

          name={FORM\_FIELDS.EMAIL}

          label="Email"

          validate={[requiredValidator, emailValidator]}

        />

        <RFields.Input

          name={FORM\_FIELDS.PASSWORD}

          label="Password"

          type="password"

          validate={[requiredValidator]}

        />

        <div className={classes.action}>

          <Button type="submit">Login</Button>

          <div className={classes.error}>{error}</div>

        </div>

      </form>

    </Page>

  );

};

const formCreator = reduxForm({

  form: FORM\_NAME,

  initialValues: {

    [FORM\_FIELDS.EMAIL]: '',

    [FORM\_FIELDS.PASSWORD]: '',

  },

});

export const Login = compose(

  withStyles(styles),

  formCreator,

)(LoginComponent);

import { ISymptom } from './symptom';

import { IMedicine } from './medicine';

import { IRecommendation } from './recommendation';

export interface IIllnessShort {

  id: number;

  name: string;

  rating?: number;

  updatedAt?: string;

  createdAt?: string;

}

export interface IIllness extends IIllnessShort {

  description: string;

  updatedAt: string;

  symptoms: ISymptom[];

  medicines: IMedicine[];

  recommendations: IRecommendation[];

}

export interface ICreateIllness extends IIllnessShort {

  symptomsIds: number[];

  medicinesIds: number[];

  symptomsIds: number[];

}

interface IIllnessReviewShort {

  review: string;

  descriptionRating: number;

  symptomsRating: number;

  medicinesRating: number;

  recommendationsRating: number;

}

export interface ICreateIllnessReview extends IIllnessReviewShort {

  medicines: ICreateIllnessMedicine[];

}

export interface ICreateIllnessMedicine {

  medicineId: number;

  rating: number;

}

export interface IIllnessReviewMedicine {

  medicineId: number;

  rating: number;

  createdAt: string;

  medicine: {

    name: string;

  };

}

export interface IIllnessReview extends IIllnessReviewShort {

  medicines: IIllnessReviewMedicine;

}

export interface IIllnessReviewGraphicData {

  date: string;

  rating: string;

}

export interface IIllnessReviewGraphicMedicines {

  id: number;

  name: string;

  ratings: IIllnessReviewGraphicData[];

}

export interface IIllnessReviewGraphic {

  name: string;

  reviews: IIllnessReviewGraphicData[];

  medicines: IIllnessReviewGraphicMedicines[];

}

export interface IMedicine {

  id: number;

  name: string;

}

export interface ISymptom {

  id: number;

  name: string;

  updatedAt?: string;

  createdAt?: string;

}

import { createStore, combineReducers, applyMiddleware, compose } from 'redux';

import thunk from 'redux-thunk';

import { reducer as form } from 'redux-form';

import { profile, IProfileState } from 'components/Profile';

import { illness, IIllness } from 'components/Illness';

import { symptom, ISymptom } from 'components/Symptom';

import { isProduction } from 'utils';

import { errors, IErrorsState } from './reducers';

export interface IStore {

  illness: IIllness,

  symptom: ISymptom,

  profile: IProfileState,

  errors: IErrorsState,

}

const composeEnhancers = isProduction ? compose : (window as any).\_\_REDUX\_DEVTOOLS\_EXTENSION\_COMPOSE\_\_ || compose;

const reducers = combineReducers( {

  illness,

  symptom,

  profile,

  errors,

  form,

});

export const store = createStore<IStore, any, any, any>(

  reducers,

  composeEnhancers(applyMiddleware(thunk)),

);

import { store, changeCodeAction } from 'store';

const commonOptions = {

  headers: {

    'Content-Type': 'application/json',

  },

};

const handleResponse = (response) =>  {

  if (response.status >= 300) {

    return Promise.reject(response.status);

  }

  return response.json()

    .catch(() => ({}));

};

const handleError = (status: number) => {

  store.dispatch(changeCodeAction(status));

  return Promise.reject(status);

};

const get = <T>(path: string, params: { [key: string]: string | number } = {}): Promise<T> => {

  const queryParams = Object.entries(params)

    .map(([k, v]) => `${k}=${v}`)

    .join('&');

  return fetch(`${path}?${queryParams}`)

    .then(handleResponse)

    .catch(handleError);

};

const post = <T>(path: string, body: object): Promise<T> => fetch(path, {

  method: 'POST',

  body: JSON.stringify(body),

  ...commonOptions,

})

  .then(handleResponse)

  .catch(handleError);

const put = <T>(path: string, body: object): Promise<T> => fetch(path, {

  method: 'PUT',

  body: JSON.stringify(body),

  ...commonOptions,

})

  .then(handleResponse)

  .catch(handleError);

const delete = <T>(path: string, body: object): Promise<T> => fetch(path, {

  method: 'DELETE',

  body: JSON.stringify(body),

  ...commonOptions,

})

  .then(handleResponse)

  .catch(handleError);

export const http = {

  get,

  post,

  put,

  delete,

};

import { AUTH\_COOKIE } from 'configs/api';

export const getCookie = (name: string) => {

  const matches = document.cookie.match(new RegExp(

    "(?:^|; )" + name.replace(/([\.$?\*|{}\(\)\[\]\\\/\+^])/g, '\\$1') + "=([^;]\*)"

  ));

  return matches ? decodeURIComponent(matches[1]) : null;

};

export const deleteCookie = (name: string): void => {

  document.cookie = `${AUTH\_COOKIE}=''; max-age=-1`;

};

export const isUserAuthorized = (): boolean => Boolean(getCookie(AUTH\_COOKIE));

export const resetUserAuthorization = (): void => deleteCookie(AUTH\_COOKIE);

export const isAllowed = (roleId: number, roles: number[]): boolean => roles.some(r => r === roleId);

import { trim } from 'lodash';

export const requiredValidator = (value: string): string => (

  trim(value)

    ? ''

    : 'Should not be empty'

);

export const emailValidator = (value: string) => (

  /^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/.test(value)

    ? ''

    : 'Enter valid email'

);

const path = require('path');

const UglifyJsPlugin = require('uglifyjs-webpack-plugin');

const HtmlWebpackPlugin = require('html-webpack-plugin');

const ForkTsCheckerWebpackPlugin = require('fork-ts-checker-webpack-plugin');

const { CleanWebpackPlugin } = require('clean-webpack-plugin');

const isDev = process.env.NODE\_ENV !== 'production';

module.exports = {

  entry: './src/index.tsx',

  output: {

    publicPath: '/',

    path: path.join(\_\_dirname, 'build'),

    filename: isDev ? '[name].js' : '[name]-[chunkhash].js',

  },

  mode: isDev ? 'development' : 'production',

  resolve: {

    modules: [

      path.join(\_\_dirname, 'node\_modules'),

      path.join(\_\_dirname, 'src'),

    ],

    extensions: ['.js', '.jsx', '.ts', '.tsx'],

  },

  devtool: isDev ? 'inline-source-map' : false,

  devServer: {

    port: 3030,

    historyApiFallback: true,

    proxy: {

      '/api': 'http://localhost:3000/',

    }

  },

  module: {

    rules: [

      {

        test: /\.(ts|tsx)$/,

        loader: 'babel-loader',

        exclude: /node\_modules/,

      },

    ],

  },

  optimization: {

    minimizer: [new UglifyJsPlugin()],

  },

  plugins: [

    new CleanWebpackPlugin(),

    new HtmlWebpackPlugin({ template: './src/index.html' }),

    new ForkTsCheckerWebpackPlugin(),

  ]

};

{

  "name": "pill.client",

  "version": "1.0.0",

  "main": "index.js",

  "author": "Uladzislau Pinchuk",

  "license": "MIT",

  "scripts": {

    "dev": "webpack-dev-server --color --progress",

    "build": "webpack",

    "build:prod": "cross-env NODE\_ENV=production webpack"

  },

  "dependencies": {

    "@material-ui/core": "4.10.0",

    "@material-ui/lab": "4.0.0-alpha.54",

    "jodit-react": "1.0.42",

    "lodash": "4.17.15",

    "react": "16.12.0",

    "react-dom": "16.12.0",

    "react-redux": "7.2.0",

    "react-router-dom": "5.1.2",

    "recharts": "1.8.5",

    "redux-form": "8.3.5",

    "redux-thunk": "2.3.0",

    "reselect": "4.0.0"

  },

  "devDependencies": {

    "@babel/cli": "7.8.4",

    "@babel/core": "7.8.4",

    "@babel/preset-env": "7.8.4",

    "@babel/preset-react": "7.8.3",

    "@babel/preset-typescript": "7.8.3",

    "@types/lodash": "4.14.149",

    "@types/node": "13.7.1",

    "@types/react": "16.9.19",

    "@types/react-dom": "16.9.5",

    "@types/react-redux": "7.1.7",

    "@types/react-router-dom": "5.1.4",

    "@types/redux-form": "8.2.3",

    "@types/redux-thunk": "2.1.0",

    "@types/reselect": "2.2.0",

    "babel-loader": "8.0.6",

    "babel-plugin-lodash": "3.3.4",

    "clean-webpack-plugin": "3.0.0",

    "cross-env": "7.0.2",

    "fork-ts-checker-webpack-plugin": "4.0.3",

    "html-webpack-plugin": "3.2.0",

    "typescript": "3.7.5",

    "uglifyjs-webpack-plugin": "2.2.0",

    "webpack": "4.41.6",

    "webpack-cli": "3.3.11",

    "webpack-dev-server": "3.10.3"

  }

}

ПРИЛОЖЕНИЕ Б

Руководство программиста

Для разработки и сопровождения данного программного продукта понадобятся:

– персональный компьютер;

– требования к ПК не ниже: 4 Гб ОЗУ, 100 Гб HDD, CPU 1,6 КГц;

– любой текстовый редактор, поддерживающий синтаксис языка TypeScript;

– сервер базы данных MySql;

– среда исполнения Node.js версией 10.6.0 и выше.

ПРИЛОЖЕНИЕ В

Руководство системного администратора

1. Требования.

– ПК, CPU 1,6 КГц и выше, ОЗУ 4 Гб и выше, HDD 100 Гб и выше;

– среда исполнения Node.js версией 10.6.0 и выше;

– сервер базы данных MySql;.

2. Установка.

– загрузить проект с удаленного репозитория;

– в дирректории pill.api поочереди ввести следующие команды: yarn install, yarn db:refresh, yarn build, yarn start.

– в дирректории pill.client поочереди ввести следующие команды: yarn install, yarn build:prod.

3. Запуск.

Введите url вида http://<your\_host>/ в поисковую строку браузера.

4. Проверка программы.

Запустите приложение. Введите в форму входа логин «login» и пароль «password». На главной странице будет находиться фильтр с симптомами, выберите хотя бы один симптом, нажмите кнопку «Search». Должна появиться таблица с заболеваниями.