

РУСЕНСКИ УНИВЕРСИТЕТ “АНГЕЛ КЪНЧЕВ”

Факултет: Природни науки и образование
Катедра: Информатика и информационни технологии

ДОПУСКАМ ДО ЗАЩИТА

Ръководител на специализираната катедра:.....
(доц. д-р Румен Русев)

**ОБЯСНИТЕЛНА ЗАПИСКА
НА ДИПЛОМЕН ПРОЕКТ**

**Тема: WEB-базирана информационна система
за обслужване на стоматологична
практика**

Дипломант:
(Д. Трифонов)

ОДОБРЯВАМ
Ръководител:.....
(гл. ас. д-р. В. Великов)

Русе, 2018 г.

РУСЕНСКИ УНИВЕРСИТЕТ “АНГЕЛ КЪНЧЕВ”

Факултет: Природни науки и образование

Катедра: Информатика и информационни технологии

РЕГИСТРИРАЛ:

Ст. инспектор:
/ М. Чолакова/

УТВЪРЖДАВАМ:

Ръководител на катедра ИИТ:
/доц. д-р Р. Русев/

ЗАДАНИЕ ЗА ДИПЛОМЕН ПРОЕКТ

на студента: Димитър Валериев Трифонов, Фак. No: 146518

Специалност: **Компютърни науки**

ОКС: **Бакалавър**

1. Тема:

WEB-базирана информационна система за обслужване на
стоматологична практика

2. Изходни данни:

Java, SWING/Java-FX, Maven, клиент-сървърни технологии, HTML, Eclipse

3. Съдържание на обяснителната записка:

- Постановка на задачата (Увод)
- Обзор на съществуващи решения. Изводи. Цел и задачи.
- Проектиране на системата, описание на използваните технологии
- Описание на програмите
- Ръководство за работа със системата
- Тестване, основни резултати. Изводи и препоръки.
- Използвана литература
- Приложения

4. Съдържание на графичната част:

- използвани технологии
- блок-схеми на основни алгоритми
- реализиран графичен интерфейс

Дата на задаване: 19.12.2017

Срок за предаване: 09.06.2018

Дипломант:
/ Д. Трифонов /

Ръководител:
/ гл. ас. д-р В. Великов /

Съдържание

1. Увод (постановка на задачата) – стр.4

2. Обзор на съществуващи решения – стр.5

- 2.1. Dentist (EMG Soft) – десктоп приложение – стр.5
- 2.2. Dentistry (Labirint 05) – десктоп приложение – стр.10
- 2.3. PracticeDent – уеб приложение – стр.12
- 2.4. Dental.bg – уеб приложение – стр.16
- 2.5. Изводи, цели и задачи – стр.17

3. Проектиране на системата – стр.18

- 3.1. Използвани технологии – стр.19
- 3.2. Архитектура на системата – стр.21

4. Описание на програмите – стр.33

- 4.1. Back-end функционалност (сървърна част) – стр.33
- 4.2. Front-end функционалност (клиентска част) – стр.51

5. Ръководство за работа със системата – стр.59

- 5.1. Системни изисквания – стр.59
- 5.2. Начин на работа (външна структура) – стр.59
- 5.3. Тестване на системата – стр.71

6. Заключение (основни резултати и изводи) – стр.74

7. Използвана литература – стр.75

1. Увод (постановка на задачата)

В днешно време софтуерът улеснява живота на хората значително. Почти няма сфера, която да не използва софтуер под някаква форма, за да се улесни дейността в нея. Например във все повече и повече хранителните магазини се ползва софтуер, който да изчислява стойността на покупката на клиент. IT фирмите ползват различни софтуерни инструменти, за да създадат НОВ софтуер! Дори тази записка е написана благодарение на софтуер за редактиране на текст. Просто светът се развива в такава насока и се очертава все повече и повече да разчитаме на информационните технологии и софтуера, за да може човечеството да се развива и да постига нови висоти.

Медицината също не прави изключение в това отношение. В днешно време медицинските лица разчитат до голяма степен на различни приложни програми (т.е. софтуер), за да улеснят работата си и също така, за да са по-ефективни и точни, когато помагат на пациентите си. Обаче съществува проблемът, че медицината има много различни области и не е добра идея да се създаде универсална програма, която обслужва всички видове „подсфери“ – обща медицина, кардиология, очни заболявания, стоматология и други. Всяка такава сфера има своята специфика – своя терминология, заболявания, специфични данни и изисквания. Именно затова по-удачно е всяка такава сфера да притежава свой собствен набор от софтуерни продукти, които да се съсредоточат ексклузивно върху проблемите в **нея**, вместо да обхващат медицината като цяло (прекалено трудна задача).

Макар в миналото най-популярни да са били „desktop приложенията“ (тези, които се инсталират на компютъра и след това се пускат на операционната система), сега все по-често започват да бъдат използвани алтернативни „мобилни приложения“ (тези, които работят на мобилни устройства като телефон, таблет и други) и „уеб приложения“ (тези, които работят в браузъра на компютъра и не изискват инсталация). Вече е много по-практично потребителят да изтегли приложение на телефона си с 2 докосвания до екрана или да отвори някой сайт, който ще свърши работата вместо него за секунди, вместо да се занимава да тегли различни неща, да проверява дали са съвместими с устройството му, да ги инсталира, да ги настройва и така нататък. Всеки вид софтуер има своите плюсове и минуси. Също така за някои задачи не е подходящо да се ползва определен тип софтуер. Например е много по-лесно да се програмира и

да се създава софтуер от десктоп или уеб приложение, отколкото от мобилно такова. Обаче ако целта на потребителя е да му бъде доставена храна до вкъщи или да си пусне музика от интернет, докато е излязъл на разходка, то едно мобилно приложение може би ще е точно това, от което се нуждае.

Естествено възниква въпросът какъв вид софтуер се изисква в медицината или по-точно „Какъв софтуер изисква една *стоматология*?“ Най-основното изискване на една стоматология е да има начин да се поддържа някаква база данни от пациенти, която съдържа подробна информация за състоянието на техните зъби. И, разбира се, трябва да има начин тази информация да бъде манипулирана и да е лесно достъпна за практикуващия стоматолог. Спорен въпрос е какъв вид приложение ще е най-подходящо за стоматология, но най-често най-доброто решение е или десктоп приложение, или уеб приложение. Причината за това е, че се очаква, че стоматологът ще ползва софтуера от офиса си и навигацията на подобно приложение ще е много по-лесна от компютър, отколкото от мобилно устройство. Плюсят на едно уеб приложение е, че потенциално предоставя възможността стоматологът да ползва приложението и от вкъщи. Десктоп приложение също може да предостави подобна функционалност, но това става по-трудно (нужна е повторна инсталация на приложението, импортиране на данни от приложението в офиса и така нататък).

Сега ще разгледаме отблизо някои съществуващи десктоп и уеб приложения, които имат за цел да предоставят решение на проблемите в управлението на една стоматология.

2. Обзор на съществуващи решения

Тъй като съществува огромно изобилие от софтуер за управляване на стоматология, долният синтезиран анализ се фокусира само върху някои продукти от **българския** пазар.

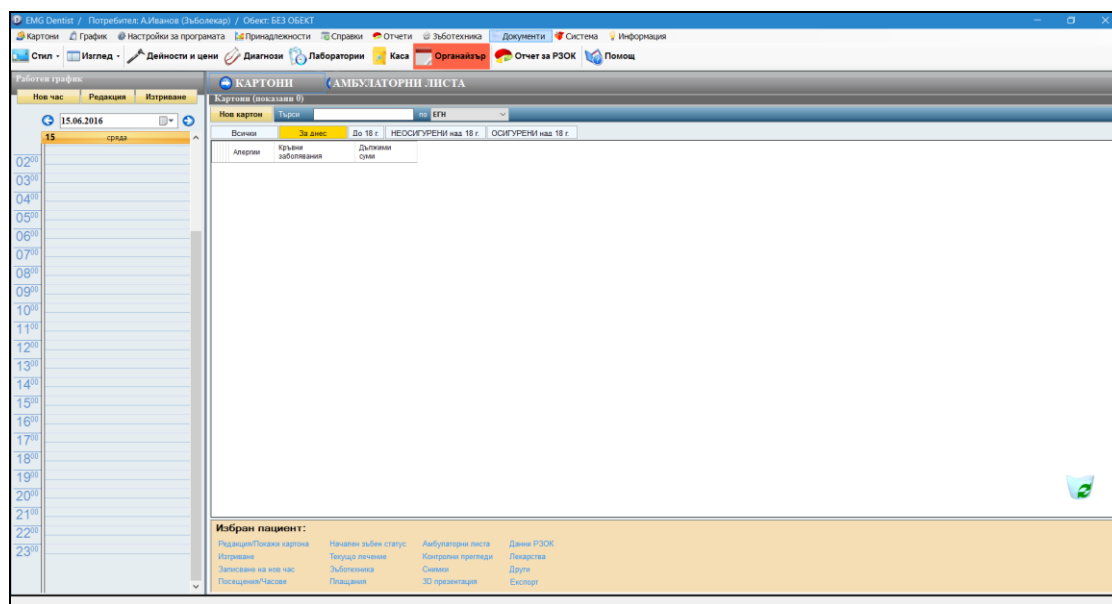
2.1. *Dentist (EMG Soft)* – десктоп приложение

Dentist е съвременен дентален софтуер. Неговите възможности обхващат почти всичко, което е необходимо за една дентална практика. В зависимост от нуждите на

всеки, Dentist е разделен на 5 версии - стандартна, НЗОК (Национална здравноосигурителна каса), сървърна, олекотена и рентген. Dentist е напълно съвместим с изискванията на НЗОК, като изготвя всичката нужна документация и отчети.

Тук ще разгледаме **стандартната** версия. Dentist (стандартната версия) включва всичко необходимо за работа и отчитане в НЗОК и работа с ЧАСТНИ ПАЦИЕНТИ плюс разширени възможности за график, лаборатории, частни дейности, частни осигурителни фондове, множество справки и отчети за финансите и много други. Dentist Standard включва на 100% и функционалностите на Dentist НЗОК.

Dentist Standard е основната версия на продуктите Dentist. [1] Тя работи на 1 компютър или мрежово с неограничен брой потребители, като е най-подходяща за отчитането на индивидуалните практики.



Фиг. 1. Органайзер в Dentist Standard

Dentist Standard може да работи в 2 режима на работа - професионален и НЗОК. Професионалният режим предоставя достъп до всички функции включени в продукта, докато НЗОК режимът предоставя само тези, които са нужни за отчитането пред Здравната каса. НЗОК режимът е подходящ за лица, които нямат особено голям опит с компютрите. НЗОК режимът предоставя начин да се попълва амбулаторният лист на бланка, която е точно копие на оригиналната такава, предоставена от НЗОК.

Професионалният режим включва всички опции на НЗОК режима, като освен работа върху амбулаторен лист, предоставя възможност за попълване на пълно досие на пациента, в което може да се проследи цялото му лечение, което е било проведено.

Може да се работи с който и да е от двата режима и дори могат да бъдат сменяни, когато потребителят пожелае. Създателите на софтуера препоръчват начинаещите потребители да започнат с НЗОК режима и в последствие да превключат към Професионален режим, за да могат да ползват всички екстри и опции на програмата.

Амбулаторен лист

Въведи "Начален зъбен статус"

Принтер

Електронно досие в НЗОК

Помощ

Затвори

Тип: **ЕГН/НСИ/Идентиф.** (Справка в НЗОК)

регистрационен номер на лечебното заведение: 202044445

код ПИД/УСИП: код ПИД/УСИП ЛГК на денталния лекар: 64 000000333

код ПИД/УСИП замест. лекар: код ПИД/УСИП замест. лекар

Име на заместителят лекар

Датум на лекар

АМБУЛАТОРЕН ЛИСТ №: / 15.06.2016 / No. ЗОК: 1

Освободен от такса: ☐ Таксованер: ☐ (провери)

АЛЕРГИЯ

Минали заболявания: Настоящи заболявания:

Имачен зъб

С кариез

Р пулпит

G периодонтит

K корен

O абсцес

E липсващ зъб

K коронка

X изкуствен зъб

Impl. Зъбен имплант

Pa парадонтоза

F фрактура

Начален статус: Текущ статус:

Маркирайте номерът на зъбът за нанасяне на статуса

Копиран в текущ

Зареди пакет с дентални данности:

Първична ДП, изцяло или частично заплащана

Въвеждане на данности

Дата: 15.06.2016 Зъб: Нов статус: Диагноза: Дълбочина: 10111 - Обстоян професионален преглед със сменяне на зъб

Въведи

Фиг. 2. Амбулаторен лист в Dentist Standard

С Dentist Standard потребителят получава всичко необходимо за отчитането на здравноосигурените пациенти, а също така и тези по частен път.

С Dentist Standard на един компютър може да работят повече от 1 зъболекар, като всеки получава индивидуален достъп до програмата и работи самостоятелно, без да вижда данните на другите потребители в системата.

Dentist Standard, както другите продукти Dentist, има система за контрол на грешките. Той не позволява въвеждането на неправилна информация за отчитането пред Здравната каса, което предпазва от допускането на неволни грешки.

Основните модули на приложението включват:

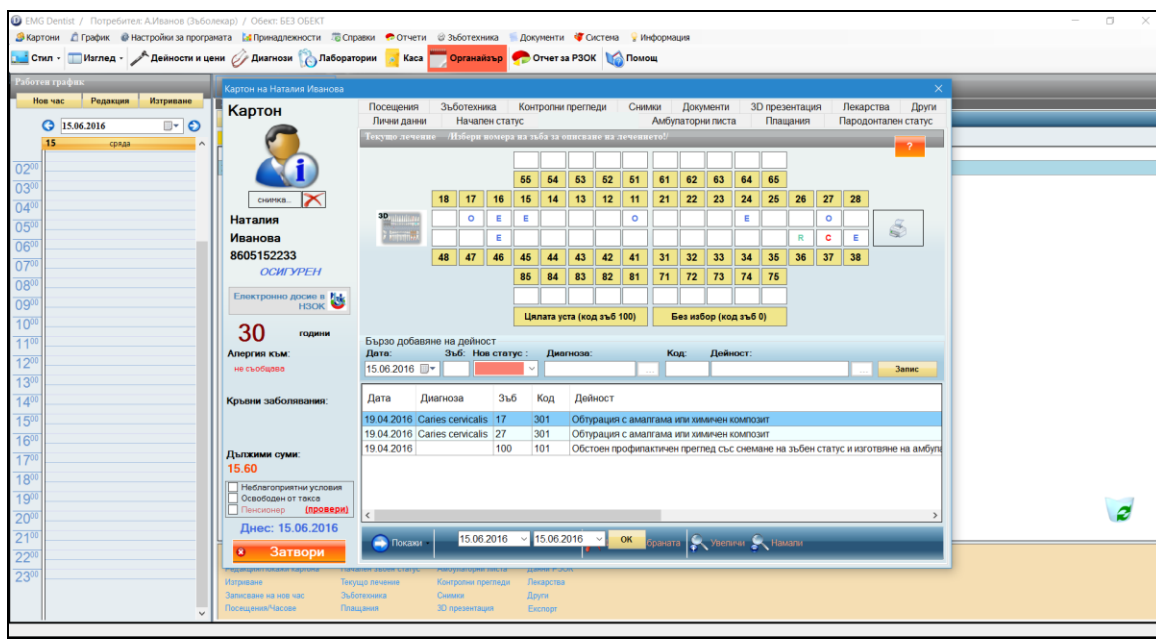
- Електронно досие в НЗОК - достъп до досието на пациента в НЗОК и виждане на извършените дейности от всички зъболекари в България
- Електронно отчитане в РЗОК (Районно здравноосигурителна каса) - всички налични функции за работа със Здравната каса и отчитане на извършените дейности. Електронни справки за здравното осигуряване и електронно досие в НЗОК. Записване на справките за здравното осигуряване в картоната на пациента и повторно разпечатване.
- График на посещенията - организиране и записване на посещенията на пациентите, контролни прегледи, подсещения на пациенти, статистика за отказали часът пациенти и др.
- Пълно досие на пациента - пазене на зъбните статуси по години, разширени възможности за лечения, следене на финансите на пациента, данни за поръчаната зъботехника, контролни прегледи, добавяне на снимки и документи, 3D зъбен статус, данни за предписани лекарства и много други. (Фиг. 2 и Фиг.3)
- Частни дейности и дейности по НЗОК - попълване на собствена номенклатура с дейности и цени
- 3D зъбен статус - допълнителен статус в триизмерен режим и детайлно отбелязване по различните повърхности.
- Органайзер - подсеждане за контролни прегледи, рожденици, поръчки за лабораторията и др. (Фиг.1)

The screenshot shows the 'Dentist Standard' software interface. The main window is titled 'Картон на Наталия Иванова' (Card of Natalia Ivanova). The interface is divided into several sections:

- Left Sidebar:** Contains a calendar view for the month of June 2016, with dates 01 to 30. Below the calendar, there is a section for '30 години' (30 years) and 'Алергия към:' (Allergy to:).
- Main Area:**
 - Top Bar:** Contains tabs for 'Посещения' (Visits), 'Зъботехника' (Dentistry), 'Контролни прегледи' (Control visits), 'Снимки' (X-rays), 'Документи' (Documents), '3D презентация' (3D presentation), 'Лекарства' (Medicines), and 'Други' (Others).
 - Personal Data:** Includes fields for 'Име' (Name), 'Презиме' (Surname), 'Фамилия' (Family name), 'Дата на раждане' (Date of birth), 'Възраст' (Age), 'Пол' (Gender), 'Гражданство' (Citizenship), and 'Код държ.' (Country code).
 - Insurance Information:** Includes fields for 'Здравно осигурен' (Insured), 'РЗОК Област' (Regional Health Insurance Fund Region), 'Здравен район' (Health district), '№ ЗОК' (Health Insurance Fund Number), 'Регистрация' (Registration), 'Населено място' (Settlement), 'П.К.' (Postal code), 'Регистрация в ЗК' (Registration in Health Insurance Fund), and 'Минали заболявания' (Previous diseases).
 - Contact Information:** Includes fields for 'Телефон' (Phone), 'Друг телефон' (Other phone), 'E-Mail', 'гр./с.' (City/Street), 'ж.к.' (Apartment), 'бп.' (Building), 'вк.' (Floor), 'ет.' (Entrance), 'ап.' (Apartment), 'ул.' (Street), 'Месторабота' (Place of work), and 'Кръвни заболявания' (Blood diseases).

Фиг. 3. Картон в Dentist Standard

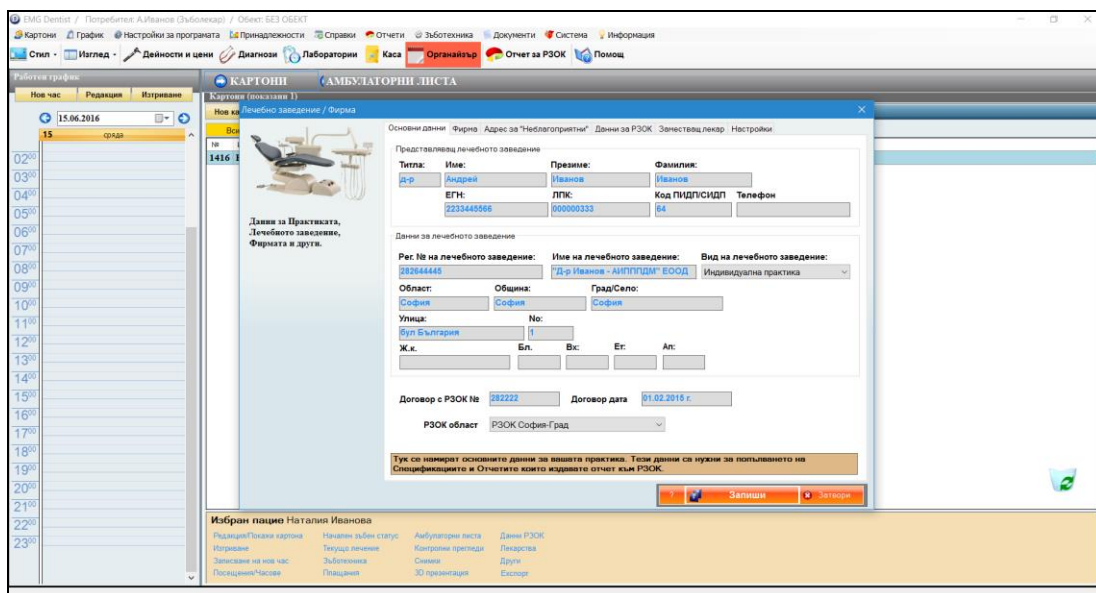
- Зъботехнически лаборатории - работа със зъботехнически лаборатории и съставяне на поръчки към тях. Следене на финансите и поръчките към лабораториите.
- Каса и финанси - следене на финансовите потоци и бързи справки за обороти, разходи и печалби.
- План на лечение - съставяне на план за лечение и изготвяне на оферта за пациента.
- Справки и Отчети - разширени функции за най-разнообразни справки и отчети.



Фиг. 4. Картон в Dentist Standard

Обобщено функциите на приложението включват:

- Пълно досие на пациента - всичката нужна информация за даден пациент на едно място
- Електронно досие в НЗОК - достъп до досието на пациента в НЗОК и виждане на извършените дейности от всички зъболекари в България
- Частни и НЗОК пациенти - може да се обработват пациенти по частен прием или по Здравната каса
- График на посещенията - отбелязване на посещенията на пациентите, контролни прегледи, подсещания и др.
- Неограничен брой потребители - с един лиценз на компютъра може да работят неограничен брой лекари по дентална медицина
- Всичко от Dentist НЗОК - всички функции от Dentist НЗОК са налични в стандартната версия



Фиг. 5. Попълване на данни за лечебно заведение в Dentist Standard

2.2. Dentistry (Labirint 05) – десктоп приложение

Стоматологичният софтуер "Dentistry" е софтуерен продукт за общопрактикуващи стоматолози и специалисти. Разработен е съгласно действащите разпоредби на НРД (Национален рамков договор) и приложенията към него. Осигурява бързина и сигурност при въвеждане на данните от преглед и изготвяне на отчет към РЗОК, както и специфични отчети за стоматолога. [2]

“Dentistry” е пример за по-примитивен софтуер, който въпреки това пак върши работа и улеснява работата на стоматолога значително.

Възможностите му включват:

- Лесен и интуитивен потребителски интерфейс, лекота и сигурност при работа
- Съхраняване и отпечатване на всички първични документи: амбулаторен лист, направление за консултация и др.
- Отпечатване на бланки, направление за хоспитализация, рецепти
- Много допълнителни бланки по желание на потребителите
- Поддръжка на пълно електронно досие за всеки пациент - лични данни, статус, извършени дейности, назначени консултации
- Лесна автоматична проверка на осигурителния статус на пациента
- Бърз достъп до вградените номенклатури – дентални дейности, дентални диагнози, населени места, кодове на изследвания

- Възможност за съхранение на графични изображения – рентгенови снимки
- Множество проверки за недопускане на грешки
- Наличие на шаблони
- Изключително лесен начин за попълване на зъбен статус - с възможност за автоматизиране
- Множество разнообразни справки
- Автоматично генериране на всички отчетни документи, залегнали в текущия НРД и изисквани от РЗОК върху хартиен и магнитен носител - спецификации, отчети, фактури и др.

Основни екрани на програмния продукт (Фиг.6):

Фиг.6. Основни екрани в Dentistry

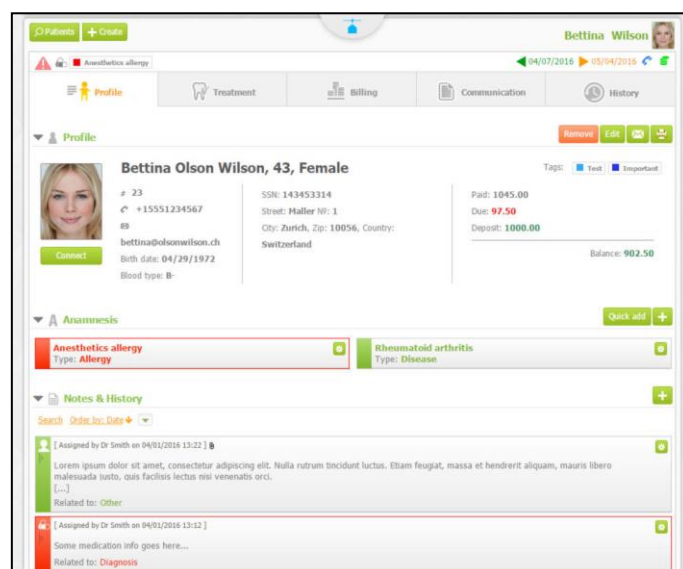
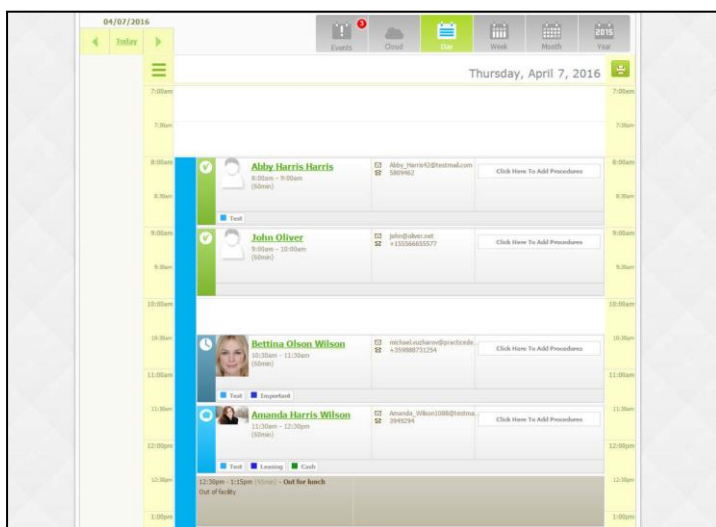
2.3. PracticeDent – веб приложение

“PracticeDent” е добър пример за качествено веб приложение, което предоставя голям набор от функции за управление на стоматология, а освен това е напълно безплатно за ползване (с до 30 регистрирани пациенти). Интерфейсът на приложението може да е както на български, така и на английски език.

Тъй като е веб базирано, приложението изисква от потребителя само да има актуален браузър като Google Chrome. Платформата е оптимизирана и за мобилни устройства с тъч-екрани, което значи, че приложението може да се ползва на таблети, смартфони и други подобни устройства. Информацията в базата данни се съхранява в центрове на Amazon, което осигурява достъп до нея през 99.99% от времето и също така осигурява резервни копия. [3]

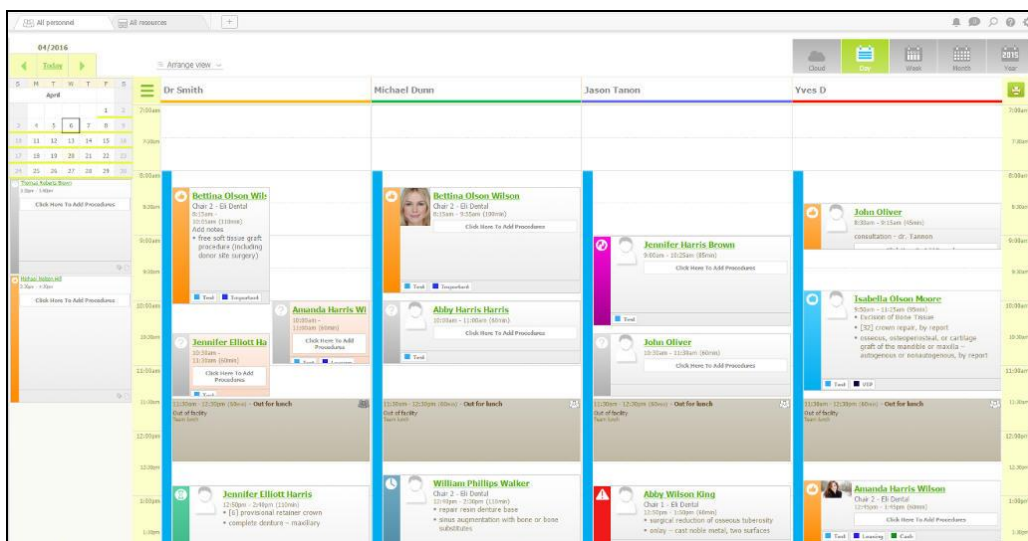
Функционалността на приложението включва:

- Календар - това е основният инструмент за планиране на времето в PracticeDent. Той е интуитивен и включва разнообразни функционалности, като например споделени събития с много участници, статуси на пациентските часове, потвърждение чрез SMS, информация за лечението, бележки и т.н. (Фиг.7)
- Електронен пациентски картон - пациентските картони съдържат демографски данни, информация за връзка, ресторативен и периодонтален статус, планове за лечение, рентгенови снимки, файлове и т.н. Своеобразен разширен профил на пациента. (Фиг.8)



Фиг.7 и 8. Календар и електронен пациентски картон в PracticeDent

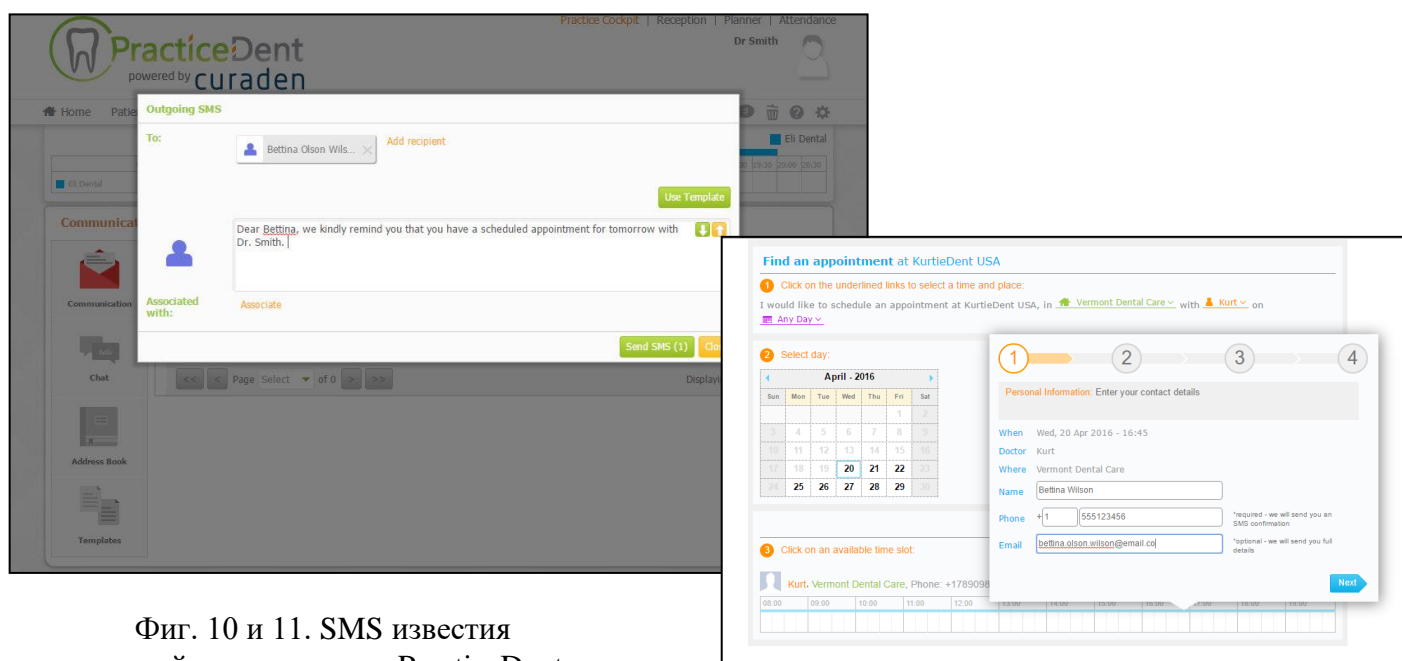
- Рецепция - този модул функционира като табло за управление на календарите и разписанията на персонала, като позволява на един човек да организира и администрира пациентските часове.



Фиг. 9. Рецепция в PracticeDent

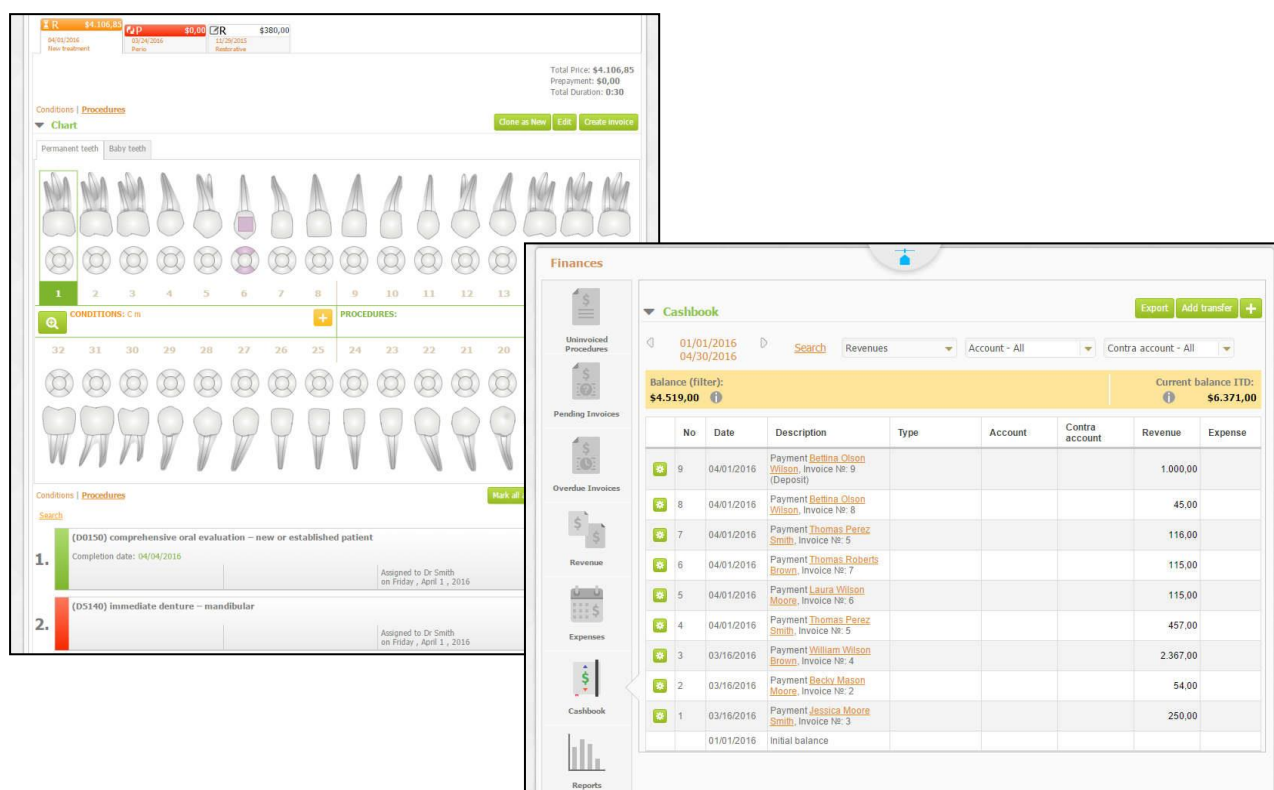
- SMS известия - PracticeDent позволява на стоматолога/потребителя да се свърже с пациентите си чрез SMS. Подсещания за час, за забавени плащания, покани за събития - всички те могат да стигнат само за секунда до пациентите. (Фиг.10)

- Онлайн резервации - този специален модул позволява на пациентите (настоящи и потенциални) да изпратят запитване за час в предпочетен от тях ден и час, като просто отворят формуляр на уебсайта на стоматолога. (Фиг.11)



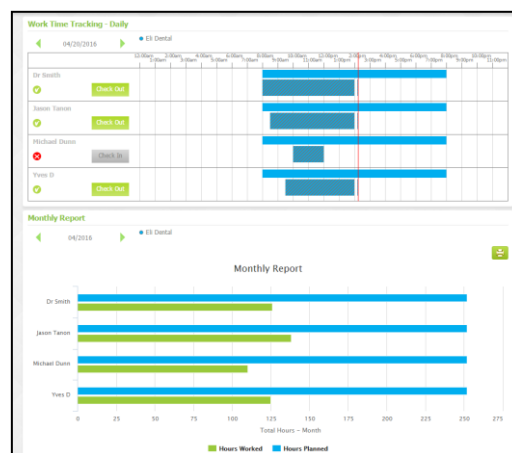
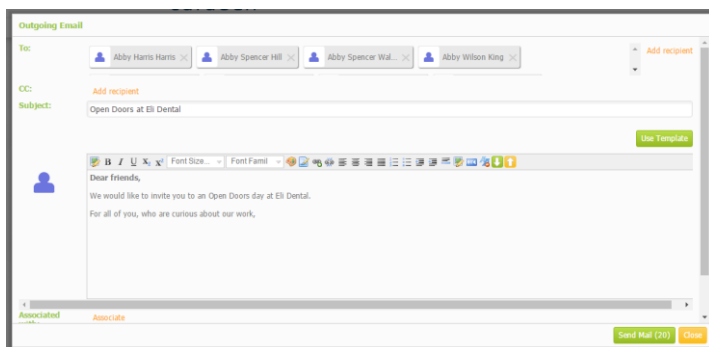
Фиг. 10 и 11. SMS известия и онлайн резервации в PracticeDent

- План за лечение - въвеждане на състояния, назначаване на манипулации, управляване и следене на план за лечение, изцяло от специален модул в пациентския профил.
- Финанси – следене на плащанията, управление на разходите и заплатите



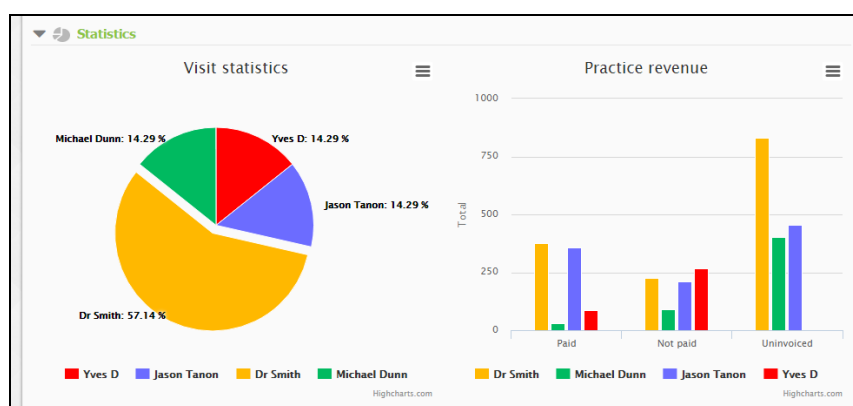
Фиг. 12 и 13. План за лечение и финанси в PracticeDent

- НЗОК интеграция - специализирана интеграция със специфичните изисквания за работа с НЗОК
- Циркулярни писма – изпращане на персонализирани послания до всеки един пациент. Полезно за имейл бюлетини, покани за събития и дори подсещания за забавени плащания.
- Модул „Присъствие“ - модул за проследяване на присъствието на персонала



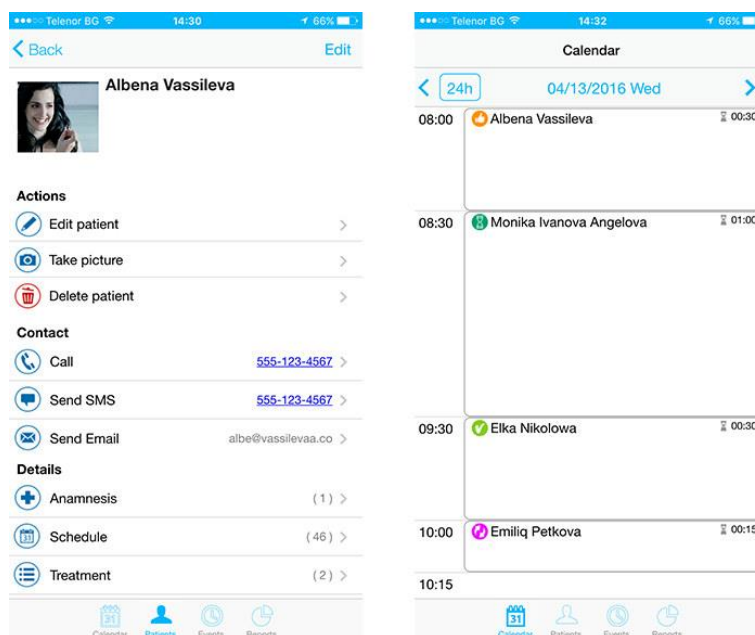
Фиг.14 и 15. Циркулярни писма и модул „Присъствие“ в PracticeDent

- Инвентар – проследяване на инвентара в различни локации, създаване на виртуални складове, определяне на аларми за намаляващи количества и дори автоматично допълване на изчерпаните артикули. Инвентарът е интегриран с електронния магазин ProductDent.
- Бизнес анализи - разнообразни отчети с визуализации и графики. Поддържа се и експортиране на суровите данни за допълнителен анализ. Бизнес модулът се обновява редовно.



Фиг. 16. Бизнес анализи в PracticeDent

- Електронен магазин - интегрираният електронен магазин ProductDent позволява лесно търсене на необходими артикули, сравняване на цени, създаване и проследяване на поръчки - от кошницата до склада.
- Мобилен достъп - PracticeDent осигурява достъп от таблети и смартфони, но също така предлага специално придружаващо приложение “Dental Go”, което може да се инсталира на Android или iOS устройство.



Фиг.17. Мобилно приложение на PracticeDent

- Неограничено съхранение на файлове – може да съхранява гигабайти информация за пациентите, като системата е гъвкава и предоставя възможност за разширение на капацитета при необходимост.
- Без нужда от ръчен бекъп – резервни копия на всичката информация в PracticeDent се пазят в три от центровете за съхранение на Amazon.
- Без допълнителен хардуер – без инсталация на вътрешни компютърни мрежи, сървъри и т.н. Нужен е само компютър, който е свързан с интернет.
- Достъп отвсякъде – може да се ползва по всяко време и отвсякъде (стига да има достъп до интернет)
- Работи навсякъде – от лаптоп, таблет, смартфон, Windows, OS X и Linux. PracticeDent изисква само актуален браузър (създателите препоръчват Google Chrome).
- Без инсталации – приложението работи в браузъра на потребителя

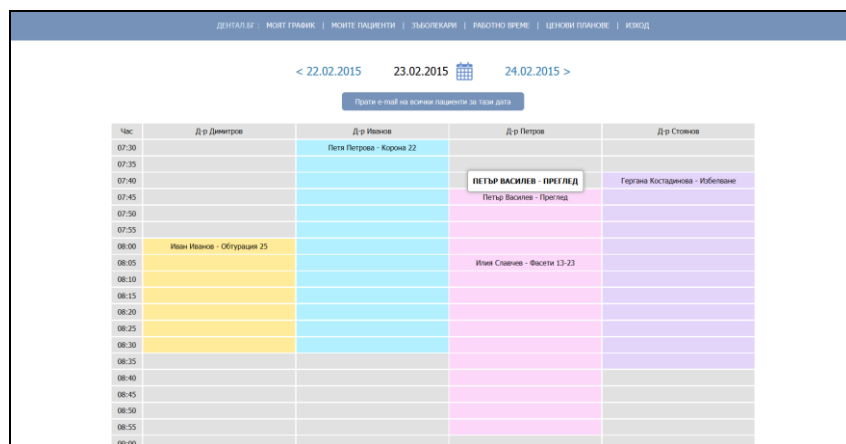
2.4. Dental.bg – уеб приложение

Dental.bg е платформа, която позволява един стоматолог да си направи профил, в който попълва информация за себе си (или за фирмата си) и това му позволява лесно да бъде открит от потенциални пациенти, а в същото време му позволява да управлява практиката си чрез вътрешното уеб приложение „Моят график“, до което се осигурява достъп след регистрация. [4] Това вътрешно уеб приложение е по-

примитивно от PracticeDent, но също предоставя добър набор от полезни функции. Приложението е напълно безплатно.

Функциите му включват:

- организация на седмичния график по часове
- добавяне на пациенти и съхранение на личните им данни и зъбния им статус
- въвеждане на клинични бележки
- изчисляване баланса на заплащанията
- изпращане на автоматичен е-mail за час при зъболекар



Фиг.18. Основни екрани на “Моят график“ в Dental.bg

2.5. Изводи, цели и задачи

На българския пазар съществуват редица решения за проблемите в управлението на една стоматология. Някои безплатни, а други платени. Някои примитивни, а други

изключително комплексни. Преобладаващите решения са десктоп приложения (много от тях не са разгледани в тази обяснителна записка, но съществуват и се използват – софтуер като Excalibur II, Хипократ D и други), но както се вижда вече постепенно навлизат в употреба уеб приложения и дори мобилни такива.

Целта на тази дипломна работа е да предостави безплатно решение за управление на една стоматология под формата на просто уеб приложение, което осигурява основната функционалност, от която се нуждае един стоматолог. Макар това да е субективно, някои потребители предпочитат прост дизайн и не желаят прекалено сложни системи с много функции, които не използват, защото това ги обърква и прави работата им по-трудна. Ако това не важи за тях, то те са свободни да изберат някое от другите приложения, които съществуват на пазара.

Обобщено задачите, които дипломната работа си поставя и решава, са следните:

- създаване на БЕЗПЛАТНО уеб приложение за управление на стоматология, което позволява да се управляват пациентите на стоматолога (личните им данни и състоянието на зъбите им, което включва и манипулациите нанесени върху всеки зъб), записаните часове на пациентите и данните на самия стоматолог, който ползва системата;
- осигуряване на прост интерфейс, който прави работата на стоматолога максимално лесна;
- осигуряване на контакт с разработчика за коригиране на грешки в системата и възможността да бъдат предлагани нови функции, които да бъдат реализирани в следващи версии на приложението
- възможност за архивиране на базата данни и за възстановяването ѝ от файл.

Приложението е насочено към стоматолози и вероятно няма да е от голяма полза на хора, които не работят в тази сфера.

3. Проектиране на системата

Дипломната работа носи заглавието „Зъболекар 2.0“, защото е базирана на десктоп приложение със същото име („Зъболекар“). „Зъболекар 2.0“ е уеб приложение и е почти идентично на десктоп варианта, но разликата е, че работи от браузъра на

потребителя и не се налага да бъде инсталиран допълнителен софтуер (достатъчно е потребителят да има модерен браузър като Google Chrome и достъп до интернет).

3.1. Използвани технологии

Системата е разработена на езика за програмиране Java (за back-end частта) и HTML/CSS/JavaScript (за front-end частта). Зъболекар 2.0 използва набор от модерни технологии, които са изброени и описани долу.

В основата на използваните в програмата **“back-end” технологии** (иначе казано сървърската част или това, което се случва „зад кулисите“) стои **Spring Framework (по-точно Spring Boot)**. Spring Framework е технологична рамка (framework) с отворен код (open source) за Java платформата. Spring предоставя много функции, които улесняват разработването на Java-базирани enterprise приложения. [5]

По-конкретно за разработването на уеб приложения се използва Spring MVC, но сега Spring Boot е даже още по-добър вариант, защото улеснява процеса на разработка значително, тъй като не се налага разработчикът сам да конфигурира всичко при създаването на проект. [6] Например при Spring MVC се налага ръчно разработчикът да попълва дълги конфигурационни файлове за всяко едно допълнение към приложението (например за конфигурация на Hibernate трябва да се пишат 20-30 реда конфигурационен код ръчно), докато Spring Boot прави всичко това автоматично. Другото голямо предимство на Spring Boot е, че позволява използването на „шаблони“ при генериране на файловете, които са нужни за даден тип проект. Например ако разработчикът иска да създаде приложение с определен тип база данни (например H2 или MySQL), то той просто трябва да маркира, че иска да използва такава и Spring Boot ще се погрижи всичко да е настроено както трябва.

Имайки това предвид, Зъболекар 2.0 използва някои такива шаблони:

- Spring Starter Web - позволява разработката на уеб приложения с вграден Tomcat сървър и Spring MVC
- Spring DevTools – позволява приложението автоматично да се обновява при всяка промяна в кода, вместо да трябва да се компилира всеки път
- Spring JPA – включва Spring Data JPA & Hibernate. Spring Data JPA позволява максимално лесно да се извлича информация от базата данни чрез Java код, а

Hibernate преобразува Java класове в таблици в базата данни (по-подробно е обяснено долу в описанието на кода на програмата)

- MySQL – базата данни, която е използвана в приложението (Spring предоставя нужния JDBC драйвер)
- Thymeleaf – “template engine”, който позволява от Spring контролер да се изпраща информация до клиентската част (тоест предоставя възможността в html файловете да може да се изведе всичката информация за пациентите, графичите и манипулациите, която е извлечена от базата данни).

Всички тези зависимости се управляват чрез **Maven** – инструмент, който автоматично изтегля нужните файлове за всяка от горните технологии.

Използваната среда за разработка е **Spring Tool Suite**, която е базирана на Eclipse, но е леко модифицирана, така че е да е максимално полезна за Spring разработчиците.

Приложението използва и модифициран код от GitHub проекта **mysql-backup4j** [7] за възстановяване или архивиране на информацията от базата данни.

Използвани са и някои **“front-end” технологии** (тоест клиентската част или това, което вижда потребителят). Голяма роля в Зъболекар 2.0 играе **jQuery** – това е популярна JavaScript библиотека, която опростява достъпа до всеки елемент от HTML страницата и позволява този елемент да бъде манипулиран по-лесно (например чрез jQuery могат да се създават елементи, да бъдат изтривани, да се променят атрибутите им, да се добавят ефекти или анимации, да се създават събития при интеракция на потребителя със страницата и т.н.) [8] Процесът на разработване става много по-лесен, когато се ползва jQuery, а не само обикновен JavaScript. Най-голямо приложение в Зъболекар 2.0 jQuery намира при зареждането на страницата и при автоматичното попълване на данните за пациент, график или манипулация. В описанието на програмата е описано как по-точно става това.

Системата използва и **Bootstrap** – това е изключително популярен framework за клиентската част, който комбинира HTML, CSS и JavaScript. Bootstrap предоставя голям набор от функции като стилизиран дизайн (елегантни бутони, таблици, форми, падащи

менюта, модални прозорци, табове и още) и анимации (с помощта на JavaScript и jQuery на фона). Освен това Bootstrap прави страниците „responsive“, тоест елементите автоматично се настройват спрямо размера на екрана и страницата не губи своя вид, когато например е отворена от смартфон. По този начин разработчикът си спестява голяма част от усилията по дизайна и по писането на код (особено усилията по писане на CSS код). Дизайнът на елементите в Зъболекар 2.0 изцяло разчита на Bootstrap.

Използвана е безплатната JavaScript библиотека **pdfmake** за генериране на PDF файл, който съдържа отчет за даден пациент. [9] В описанието на програмата е обяснено как по-точно се случва това.

Може би най-незначителната (но все пак допринасяща с малко) използвана библиотека е **Emoji CSS**, която позволява лесно вмъкване на различни емоѝ изображения, за да може прозорците за грешка в приложението да изглеждат по-дружелюбни към потребителя (т.е. да са „user-friendly“).

За **тестване** на системата е използвано Windows приложението **XAMPP** (v.3.2.2), което стартира MySQL и позволява създаването на база данни от Spring. XAMPP също стартира и *Apache* сървър, за да има достъп до *phpMyAdmin* контролния панел, с който по-лесно може да се следи състоянието на базата данни. Системата е тествана на последните версии на Google Chrome, Mozilla Firefox, Opera, Edge и Internet Explorer. Графичният инструмент **Jeddict** е използван за генериране на диаграма от JPA-анотиран Java код, която описва връзките между различните единици в системата (тоест ERD – Entity Relationship Diagram). Тази диаграма е показана в следващата част от обяснителната записка на дипломната работа.

3.2. Архитектура на системата

Тук ще бъде разгледана **вътрешната** структура на системата, а външната структура (това, което вижда потребителят) ще бъде обяснена в частта „Ръководство за работа със системата“ на обяснителната записка.

Вътрешно системата се реализира благодарение на 5 вида „единици“ (entities), които имат различни връзки помежду си. Всяка единица има свои атрибути (свойства) и е изразена под формата на Java клас. На база на тези единици е изградена база данни, чиито таблици автоматично се генерират от Hibernate.

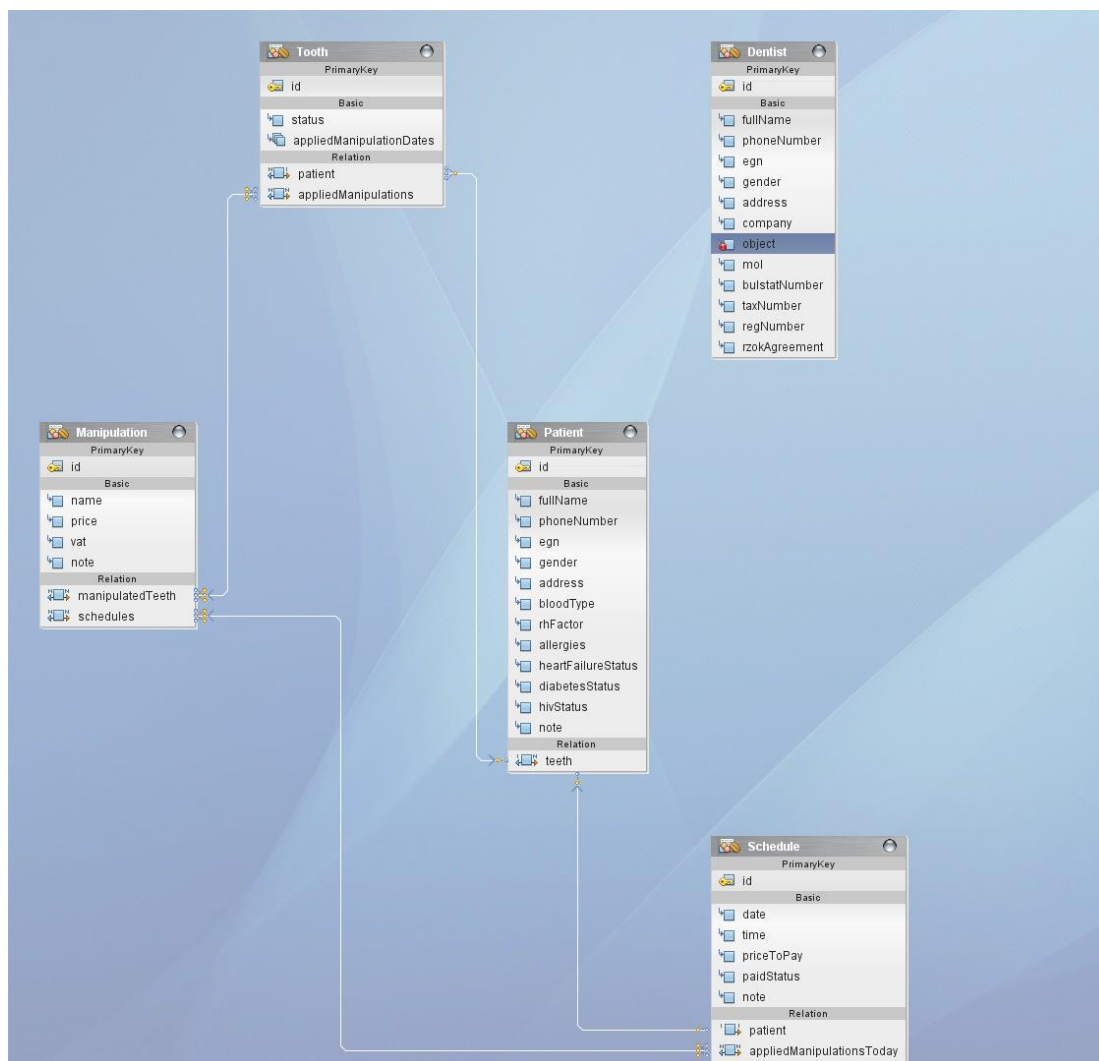
Единиците са:

- Зъболекар (**Dentist**)
- Манипулация (**Manipulation**)
- Пациент (**Patient**)
- Зъб (**Tooth**)
- График (**Schedule**)

На долната схема са показани връзките между различните единици. Единствено **Dentist** няма никаква връзка с другите единици и е „изолирана“, защото на този етап уеб приложението позволява да бъде ползвано само от един стоматолог и информацията за стоматолога не участва в останалата работа на системата.

Всяка единица има свой ключов атрибут „id“ (номер), за да може всеки нейн представител (например всеки зъб, всеки пациент, всяка манипулация и така нататък) да може да бъде лесно открит в базата данни.

Под схемата са описани всички единици, атрибутите им и връзките между тях.



Фиг.19. ERD на системата

Описание на атрибутите на единицата **Dentist** (данните за стоматолога):

Име	Тип	Описание	Забележка
id	Integer / цяло число	Уникален номер на стоматолога, който ползва системата	Тъй като системата на този етап поддържа работата само на един стоматолог, този номер винаги е равен на 1.
fullName	String / символен низ	Цяло име на стоматолога	
phoneNumber	String / символен	Телефон на	Макар телефонният

	низ	стоматолога	номер да представлява число, е по-практично да се съхранява като символен низ.
egn	String / символен низ	ЕГН на стоматолога	Макар ЕГН-то да представлява число, е по-практично да се съхранява като символен низ.
gender	String / символен низ	Пол на стоматолога	Стойността може да бъде "male" (мъж) или „female" (жена) и бива подадена от формата, която изпраща данните.
address	String / символен низ	Адрес на стоматолога	
company	String / символен низ	Фирма на стоматолога	
object	String / символен низ	Обект	
mol	String / символен низ	МОЛ – материално отговорно лице	
bulstatNumber	String / символен низ	Булстат номер	
taxNumber	String / символен низ	Данъчен номер	
regNumber	String / символен низ	Регистрационен номер на лечебното заведение	
rzokAgreement	String / символен	Договор с РЗОК	

	НИЗ		
--	-----	--	--

(Таблица 1. Атрибути на единицата Dentist)

Описание на атрибутите на единицата **Manipulation** (данни за манипулация):

Име	Тип	Описание	Забележка
id	Integer / цяло число	Уникален номер на манипулацията	
name	String / символен низ	Уникално име на манипулацията	Не е позволено да съществуват манипулации с еднакви имена.
price	Double / реално число	Цена на манипулацията	Стойността е в лева.
vat	Double / реално число	ДДС	Стойността е в проценти.
note	String / символен низ	Забележка за манипулацията	
manipulatedTeeth	List<Tooth> / списък от зъби	Списък от зъби на пациенти, на които е приложена тази манипулация	Тъй като една манипулация може да е поставена върху много зъби, а в същото време върху един зъб могат да са поставени много манипулации, между атрибутите „manipulatedTeeth” (от Manipulation) и „appliedManipulations” (от Tooth) има връзка many-to-many (много към много).
schedules	List<Schedule> / списък от	Списък от графици, на които	Тъй като една манипулация може да е

	графици	е била приложена тази манипулация (тоест на кои дати)	поставена на много графици (на много записани часове), а в същото време на един график (на един записан час) могат да бъдат поставени много манипулации, между атрибутите „schedules” (от Manipulation) и „appliedManipulationsToday” (от Schedule) съществува връзка many-to-many (много към много).
--	---------	---	---

(Таблица 2. Атрибути на единицата Manipulation)

Описание на атрибутите на единицата **Patient** (данни за пациент):

Име	Тип	Описание	Забележка
id	Long / цяло число	Уникален номер на пациента	
fullName	String / символен низ	Цяло име на пациента	Името е задължително поле.
phoneNumber	String / символен низ	Телефон на пациента	Макар телефонният номер да представлява число, е по-практично да се съхранява като символен низ.
egn	String / символен низ	ЕГН на пациента	Макар ЕГН-то да представлява число, е по-

			<p>практично да се съхранява като символен низ.</p> <p>ЕГН-то е задължително поле и не може да се повтаря.</p>
gender	String / символен низ	Пол на пациента	Стойността може да бъде "male" (мъж) или "female" (жена) и бива подадена от формата, която изпраща данните.
address	String / символен низ	Адрес на пациента	
bloodType	String / символен низ	Кръвна група на пациента	Стойностите могат да са „0“, „А“, „В“ и „AB“.
rhFactor	String / символен низ	Резус фактор	Стойностите могат да са „+“ и „-“.
allergies	String / символен низ	Алергии на пациента	
heartFailureStatus	String / символен низ	Статус на сърдечна недостатъчност (има ли такава или не)	Стойностите могат да са NULL или "on". Този атрибут се контролира от checkbox във формата за подаване на данните – когато не е маркиран, стойността му става NULL, а когато е,

			става "on".
diabetesStatus	String / символен низ	Статус на диабет (има ли такъв или не)	Стойностите могат да са NULL или "on". Този атрибут се контролира от checkbox във формата за подаване на данните – когато не е маркиран, стойността му става NULL, а когато е, става "on".
hivStatus	String / символен низ	Статус на ХИВ (ХИВ позитивен ли е пациентът или не)	Стойностите могат да са NULL или "on". Този атрибут се контролира от checkbox във формата за подаване на данните – когато не е маркиран, стойността му става NULL, а когато е, става "on".
note	String / символен низ	Забележка	
teeth	List<Tooth> / списък от зъби	Списък със зъбите на пациента	Един пациент може да има много зъби, а всеки зъб принадлежи само на един пациент. Затова съществува връзка one-to-many

			(едно към много) между атрибутите „patient“ (от Tooth) и “teeth” (от Patient) и many-to-one (много към едно) между “teeth” и “patient”.
--	--	--	---

(Таблица 3. Атрибути на единицата Patient)

Описание на атрибутите на единицата **Tooth** (данни за зъб на пациент):

Име	Тип	Описание	Забележка
id	Long / цяло число	Уникален номер на зъба	Зъбите на първия пациент ще са номерирани от 1 до 32, на следващия от 33 до 64 и т.н.
status	String / символен низ	Състояние на зъба	Позволените стойности "Здрав", "Кариес", "Пулпит", "Гангрена", "Корен", "Обтурация", "Липсващ зъб", "Коронка", "Изкуствен зъб", "Фрактура", "Пародонтоза I", "Пародонтоза II", "Пародонтоза III", "Периодонтит", "Пломба"
appliedManipulationDates	List<String> / списък от	Списък от датите, на които	Тези стойности се подават, когато се

	символни низове	са поставени манипулациите върху този зъб	създаде график.
patient	Patient / пациент	Пациент, на когото принадлежи този зъб	<p>Всеки зъб принадлежи на един пациент, а един пациент може да има много зъби. Затова съществува връзка one-to-many (едно към много) между атрибутите „patient“ (от Tooth) и “teeth” (от Patient) и many-to-one (много към едно) между “teeth” и “patient”.</p> <p>При създаването на пациент, всеки негов зъб получава този пациент като стойност за атрибута “patient”.</p>
appliedManipulations	List<Manipulation> / списък от манипулации	Списък от манипулации, които са поставени върху този зъб	<p>Тези стойности се подават, когато се постави манипулация върху зъба чрез някой график.</p> <p>Тъй като една манипулация може да е поставена върху много зъби, а в</p>

			същото време върху един зъб могат да са поставени много манипулации, между атрибутите „manipulatedTeeth” (от Manipulation) и „appliedManipulations” (от Tooth) има връзка many-to-many (много към много).
--	--	--	---

(Таблица 4. Атрибути на единицата Tooth)

Описание на атрибутите на единицата **Schedule** (данни за график):

Име	Тип	Описание	Забележка
id	Long / цяло число	Уникален номер на графика	
date	String / символен низ	Дата на записания час	Комбинацията дата/час трябва да е уникална.
time	String / символен низ	Записан час	Комбинацията дата/час трябва да е уникална.
priceToPay	Double / реално число	Цената, която трябва да бъде заплатена от пациента	Стойността се базира на общата цена на поставените манипулации върху зъбите на пациента на този ден/час (с включено ДДС)
paidStatus	String / символен низ	Статус на плащането (платил ли	Стойностите могат да са NULL или “on”. Този атрибут се контролира от

		си е пациентът за извършените услуги или не)	checkbox във формата за подаване на данните – когато не е маркиран, стойността му става NULL, а когато е, става “on”.
note	String / символен низ	Забележка	
patient	Patient / пациент	Пациент, който ще бъде (или вече е бил) лекуван на този ден/час.	Всеки записан час (всеки график) изисква да има пациент, който ще бъде лекуван тогава, но не всеки пациент задължително има записан час. Затова има едностранна връзка one-to-one (едно към едно) между Schedule и Patient, изразена чрез този атрибут.
appliedManipulationsToday	List<Manipulation> / списък с манипулации	Списък с извършвани манипулации на тази дата/час	Тъй като една манипулация може да е поставена на много графици (на много записани часове), а в същото време на един график (на един записан час) могат да бъдат поставени много манипулации, между атрибутите „schedules” (от Manipulation) и „appliedManipulationsToday” (от Schedule) съществува връзка many-to-many

			(много към много).
--	--	--	--------------------

(Таблица 5. Атрибути на единицата Schedule)

4. Описание на програмите

За да е възможно най-прегледно, тук файл по файл ще бъде обяснено как точно работи приложението. Ще бъде разгледана както back-end частта (сървърната част), така и front-end частта (клиентската част).

4.1. Back-end функционалност (сървърна част)

Пакет: net.dvt32.DentistManager

Файл: **DentistManagerApplication.java**

Описание: автоматично генериран от Spring Boot клас, чрез който приложението се зарежда през ApplicationContext класа на Spring.

Файл: **ServletInitializer.java**

Описание: автоматично генериран от Spring Boot конфигурационен клас, който подготвя приложението за стартиране.

Файл: **application.properties**

Описание: файл, който съдържа настройки на Spring за приложението.

Настройките са:

- spring.jpa.hibernate.ddl-auto – настройка на Hibernate за генерираните таблици в базата данни, която диктува какво ще се случи с тях, когато разработчикът спре приложението. "update" казва, че таблиците няма да бъдат изтрети, а „create-drop” ще ги изтрие при спиране на приложението и ще ги генерира отново при следващото стартиране (компилиране).

- *spring.datasource.url* – това е JDBC адреса, чрез който се прави връзка към базата данни. Всеки тип база данни има специфичен формат за този адрес, който е определен в документацията ѝ (този е валиден само за MySQL). В случая „dentistmanager_db” е името на предварително създадената база данни. Използва се параметъра „characterEncoding=UTF-8”, за да може стойностите на кирилица да се показват правилно в таблиците.
- *spring.datasource.username* – това е потребителското име, с което се осъществява достъп до базата данни. В случая се очаква, че това потребителско име е „root”.
- *spring.datasource.password* – това е паролата, с която се осъществява достъп до базата данни. В случая се очаква, че няма парола.
- *spring.jpa.properties.hibernate.dialect* – диалект за MySQL. В случая се ползва MySQL5 диалект.
- *server.error.whitelabel.enabled* – чрез тази настройка са забранени съобщенията за грешка по подразбиране от Spring (вместо това са използвани собствени, които са обяснени надолу)

Файл: **pom.xml**

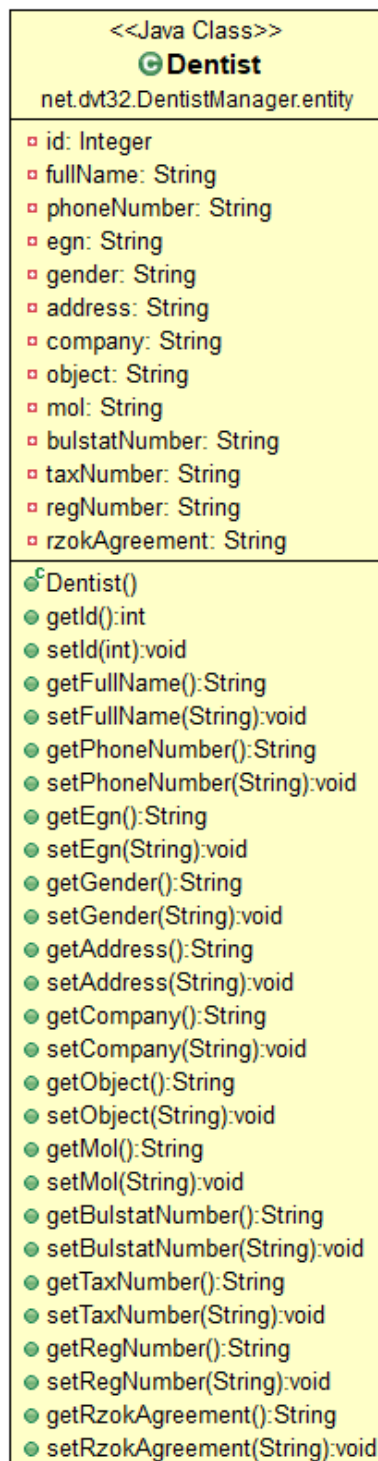
Описание: т.нар. “project object model”, в който са описани нужните зависимости, които Maven да изтегли (Spring Data JPA, Hibernate, Thymeleaf, MySQL драйвера и всички останали), както и други данни за проекта (име, описание и други).

Пакет: net.dvt32.DentistManager.entity

Файл: **Dentist.java**

Описание:

Клас, който представлява „единица“ (entity) в системата, която съдържа данните за стоматолога. Hibernate вижда специалната JPA анотация @Entity и разбира, че трябва да го превърне в таблица в посочената в настройките (*application.properties* файла) база данни. За да стане това, обаче, трябва да имаме първичен ключ и в случая това е атрибутът “id”, който е означен с анотацията @Id. Hibernate вижда анотацията и създава таблица с първичен ключ “id” и с останалите атрибути, които са от тип String (в базата данни биват представени като VARCHAR).



Фиг.20. Клас диаграма на Dentist

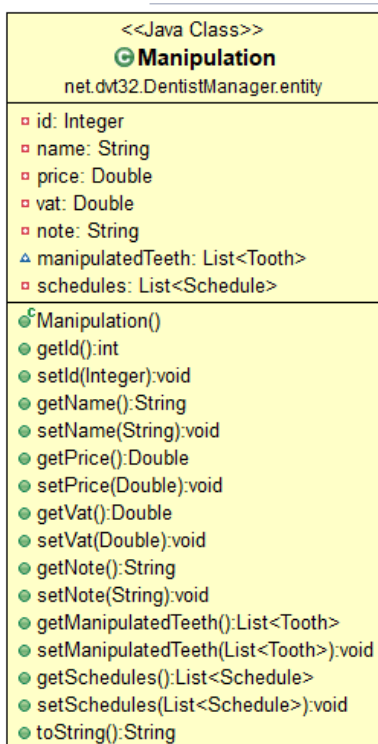
Важно е да се отбележи, че е нужен **конструктор без параметри**, за да може Hibernate да си свърши работата (за да може да инициализира обектите чрез Java рефлексия). [10] По принцип няма нужда обектите ръчно да бъдат инициализирани, но в този случай са, за да може Thymeleaf да покаже някакви стойности във формата за попълване на данните на стоматолога, когато все още не са зададени такива (това ще е обяснено по-подробно в анализа на “settings.html”).

Всички методи в този клас са просто обикновени getters/setters, които се използват за манипулиране и извличане на данните.

Файл: **Manipulation.java**

Описание:

Клас, който представлява „единица“ (entity) в системата, която съдържа данните за една манипулация. Hibernate вижда специалната JPA анотация @Entity и разбира, че трябва да го превърне в таблица в посочената в настройките (*application.properties* файла) база данни. За да стане това, обаче, трябва да имаме първичен ключ и в случая това е атрибутът “id”, който е означен с анотацията @Id. Hibernate вижда анотацията и създава таблица с първичен ключ “id”.



Фиг.21. Клас диаграма на Manipulation

Особеното в случая е, че е използвана т.нар. „стратегия за генериране“ `IDENTITY` с анотацията `@GeneratedValue(strategy = GenerationType.IDENTITY)`, която определя на какъв принцип ще бъде генериран номерът за ключа (по подразбиране стратегията за генериране е `AUTO`, но в този случай е `IDENTITY` и означава, че ключовете ще бъдат номерирани от 1 нататък).

Полето "name" има анотацията @NotEmpty, която означава, че манипулацията няма да бъде създадена, ако стойността е празна или null (полето задължително трябва да има някаква валидна стойност). Отделно има и анотацията @Column(unique=true), която се грижи да няма манипулации с повтарящи се имена (ако такъв случай настъпи, ще се появи съобщение за грешка).

Полето "manipulatedTeeth" има анотация @ManyToMany, която създава връзка „много към много“ с атрибута „appliedManipulations“ от Tooth класа (тоест една манипулация може да е била поставена върху много зъби, а на един зъб може да има поставени много манипулации). Анотацията @JsonIgnore премахва този атрибут от JSON сериализацията/десериализацията на Manipulation обекта. Аналогично и за атрибута „schedules“.

Отново имаме конструктор без параметри (за Hibernate) и getter/setter методи. Включена е и имплементация на toString(), която се ползва само от разработчика за по-лесно откриване на бъгове.

Файл: **Patient.java**

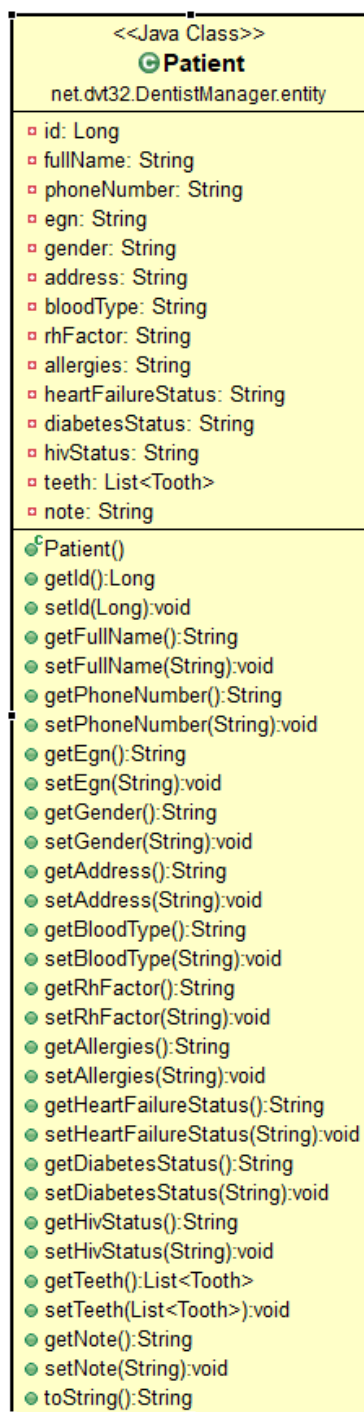
Описание:

Клас, който представлява „единица“ (entity) в системата, която съдържа данните за един пациент. Hibernate вижда специалната JPA анотация @Entity и разбира, че трябва да го превърне в таблица в посочената в настройките (*application.properties* файла) база данни. За да стане това, обаче, трябва да имаме първичен ключ и в случая това е атрибутът "id", който е означен с анотацията @Id. Hibernate вижда анотацията и създава таблица с първичен ключ "id".

Отново се използва стратегията за генериране на ключ IDENTITY, което означава, че пациентите са номерирани от 1 нататък (в контекста на MySQL това се равнява на AUTOINCREMENT [11]).

Полето "fullName" има анотация @NotEmpty, която не позволява то да има null стойност или да е празно. Същото важи и за полето "egn", но то има и

@Column(unique=true) анотация, която спира вмъкването на пациенти с повтарящи се ЕГН-та (номерът трябва да е уникален за всеки пациент).



Фиг.22. Клас диаграма на Patient

По-особено е полето „teeth”, което има анотация @OneToMany. Чрез тази анотация се определя връзка „едно към много“ с полето “patient” от Tooth (тоест един пациент има много зъби, а всеки зъб принадлежи на точно един пациент) и сега в таблицата „tooth” в базата данни за всеки зъб ще има атрибут “patient_id” със стойност id-то на пациента,

на когото принадлежи този зъб. Благодарение на частта „cascade = CascadeType.ALL”, когато един пациент е изтрит или обновен, зъбите му също ще бъдат изтрети или обновени.

Отново има конструктор без параметри, за да може Hibernate да инициализира обекта успешно.

Останалите методи са просто обикновени getters/setters, а последният е имплементация на toString(), която улеснява откриването на проблеми в програмата от разработчика.

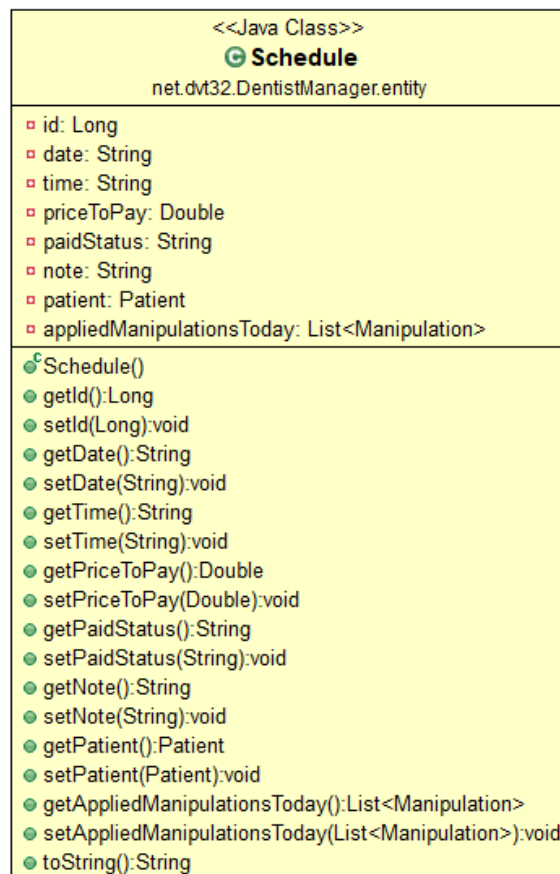
Файл: **Schedule.java**

Описание:

Клас, който представлява „единица“ (entity) в системата, която съдържа данните за един график (един записан час). Hibernate вижда специалната JPA анотация @Entity и разбира, че трябва да го превърне в таблица в посочената в настройките (application.properties файла) база данни. За да стане това, обаче, трябва да имаме първичен ключ и в случая това е атрибутът “id”, който е означен с анотацията @Id. Hibernate вижда анотацията и създава таблица с първичен ключ “id”.

В началото е използвана анотацията @Table, с която се определят някои настройки за вътрешното представяне на таблицата в базата данни. Използван е атрибута “uniqueConstraints” върху „schedule_date” и „schedule_time”, което означава, че комбинацията от дата/час трябва да е уникална, иначе записаният час не е валиден и няма да бъде добавен в базата данни. Друго важно нещо в случая е, че стойностите за uniqueConstraints са имена, които са определени в @Column анотация (обяснено долу), а не имената на полетата, които са декларирани в Java класа.

Интересно е, че полетата „date” и „time” имат анотация @Column (която определя настройки за вътрешното представяне на полето в базата данни) и име, което се различава от това на променливите в класа (не е “date”, ами „schedule_date” и същото за “time” и “schedule_time”). Това е така, защото “date” и „time” са резервирани думи в MySQL. Полетата имат също и дължина, която е определена чрез атрибута “length” на анотацията @Column.



Фиг.23. Клас диаграма на Schedule

Полето „priceToPay” е предварително инициализирано с 0, за да се избегнат потенциални грешки, в които някой метод се опитва добави към стойността на полето или да извади от нея.

Над полето “patient” има анотация @OneToOne, която определя връзка „едно към едно“. Тоест един график има един пациент (записаният час е за точно един пациент).

Полето „appliedManipulationsToday” има анотация @ManyToMany, която определя връзка „много към много“. Тоест на един график (записан час) могат да бъдат извършени много манипулации, а една манипулация може да бъде извършена в различни дни/часове. Атрибутът „cascade” в анотацията е нужен, за да може при изтриване на даден записан час, да не бъдат изтрети и манипулациите, които са били извършени тогава (затова стойността му не е CascadeType.ALL, а вместо това „{CascadeType.PERSIST, CascadeType.REFRESH, CascadeType.MERGE}“ – липсва CascadeType.REMOVE). Чрез анотацията @JoinTable се създава трета таблица в базата данни, която илюстрира many-to-many връзката между манипулациите и графициите.

Съществува конструктор без параметри, за да може Hibernate да инициализира обектите. Останалите методи са getters/setters и има toString() имплементация за по-лесно откриване на грешки от разработчика.

Файл: **Tooth.java**

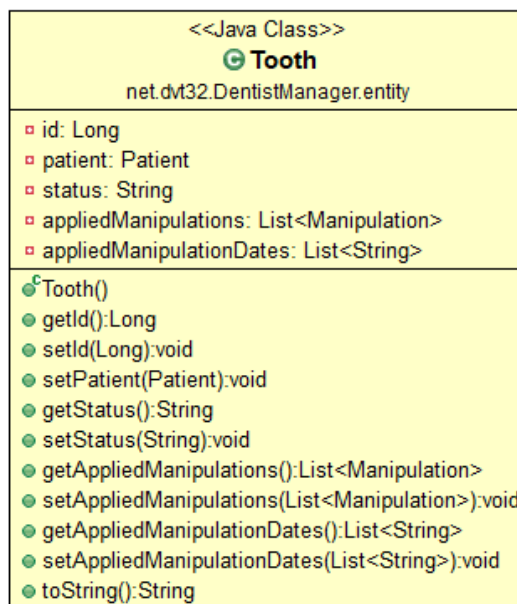
Описание:

Клас, който представлява „единица“ (entity) в системата, която съдържа данните за един зъб на пациент. Hibernate вижда специалната JPA анотация @Entity и разбира, че трябва да го превърне в таблица в посочената в настройките (*application.properties* файла) база данни. За да стане това, обаче, трябва да имаме първичен ключ и в случая това е атрибутът “id”, който е означен с анотацията @Id. Hibernate вижда анотацията и създава таблица с първичен ключ “id”.

Отново се използва стратегията за генериране на ключ IDENTITY, което означава, че пациентите са номерирани от 1 нататък. Първият пациент ще има зъби номерирани от 1 до 32, вторият от 33 до 64 и така нататък.

Полето „patient” е анотирано с @ManyToOne, а това определя връзка „много към едно“ (тоест много зъби принадлежат на един пациент и всеки зъб има точно един пациент, на когото принадлежи). Анотацията @JoinColumn(name=“patient_id”) определя, че Tooth е „притежателят“ на връзката [12] и колоната, която показва връзката между единиците, ще се казва “patient_id” (тоест в таблицата “tooth” ще има поле “patient_id”, което показва на кой пациент принадлежи даден зъб).

По-надолу има поле „appliedManipulations” с @ManyToMany анотация, която изразява връзка „много към много“ (на един зъб може да се поставят много манипулации, както и една манипулация може да бъде поставена върху много зъби). Добавен е и атрибут “cascade = {CascadeType.PERSIST, CascadeType.REFRESH, CascadeType.MERGE}”, за да може при изтриване на зъб (тоест при изтриване на пациент, тъй като индивидуален зъб не може да бъде изтрит) да не се изтрива и манипулацията, която е била приложена върху него.



Фиг.24. Клас диаграма на Tooth

Интересно е полето „appliedManipulationDates”, което има анотация `@ElementCollection`. Чрез тази анотация се създава таблица “tooth_applied_manipulation_dates”, която съхранява датите, на които е извършвана манипулация върху даден зъб. Тази информация се добавя при създаването на график, в който се извършва манипулация върху определен зъб.

Останалите методи са просто getters/setters за полетата. Накрая има имплементация на `toString()`, която улеснява разработчика в откриването на грешки в програмата. Има и конструктор без параметри, който се изисква от Hibernate за инициализиране на обектите.

Пакет: `net.dvt32.DentistManager.repository`

Файл: **DentistRepository.java**

Описание:

Интерфейс, който реализира функционалността на Spring Data JPA. Механизмът е следния – създава се интерфейс, който наследява `CrudRepository` (CRUD е съкращение за “Create, Read, Update, Delete”, тоест основните операции с данните в

една таблица) и първият параметър на `CrudRepository` е единицата, за която ще се извлича информация от базата данни (в случая `Dentist`). Вторият параметър е типът на първичния ключ на единицата (в случая `Integer`).

Сега може да се създаде обект от тип `DentistRepository` и чрез него да се извлича информация от базата данни – например методът `findAll()` ще върне всички стоматолози в базата данни, а `save()` ще обнови данните за някой стоматолог. [13] Не се налага разработчикът да настройва нищо друго, Spring Data JPA ще свърши работата вместо него.

Файл: **ManipulationRepository.java**

Описание:

Интерфейс, който реализира функционалността на Spring Data JPA. Механизмът е следния – създава се интерфейс, който наследява `CrudRepository` (CRUD е съкращение за “Create, Read, Update, Delete”, тоест основните операции с данните в една таблица) и първият параметър на `CrudRepository` е единицата, за която ще се извлича информация от базата данни (в случая `Manipulation`). Вторият параметър е типът на първичния ключ на единицата (в случая `Integer`).

Сега може да се създаде обект от тип `ManipulationRepository` и чрез него да се извлича информация от базата данни – например методът `findById()` ще върне манипулация с подаден ключ (ако съществува такава), а `deleteById()` ще изтрие манипулация с подадено id (отново само ако съществува). Не се налага разработчикът да настройва нищо друго, Spring Data JPA ще свърши работата вместо него.

Файл: **PatientRepository.java**

Описание:

Интерфейс, който реализира функционалността на Spring Data JPA. Механизмът е следния – създава се интерфейс, който наследява `CrudRepository` (CRUD е съкращение за “Create, Read, Update, Delete”, тоест основните операции с данните в една таблица) и първият параметър на `CrudRepository` е единицата, за която ще се извлича информация от базата данни (в случая `Patient`). Вторият параметър е типът на първичния ключ на единицата (в случая `Long`).

Сега може да се създаде обект от тип `PatientRepository` и чрез него да се извлича информация от базата данни – например методът `count()` ще върне броя пациенти в базата данни, а `deleteAll()` ще изтрие информацията за всички пациенти от там. Не се налага разработчикът да настройва нищо друго, Spring Data JPA ще свърши работата вместо него.

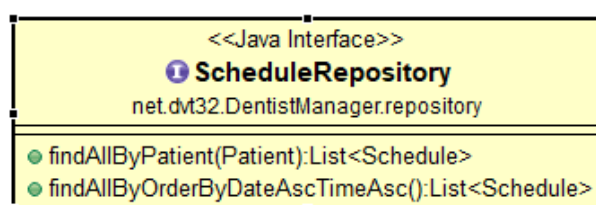
Забележка: Не се налага да се създава хранилище за `Tooth` (тоест интерфейс `ToothRepository`), защото зъбите се създават при създаването на пациент, а пък това става именно с `PatientRepository`.

Файл: **ScheduleRepository.java**

Описание:

Интерфейс, който реализира функционалността на Spring Data JPA. Механизмът е следния – създава се интерфейс, който наследява `CrudRepository` (CRUD е съкращение за “Create, Read, Update, Delete”, тоест основните операции с данните в една таблица) и първият параметър на `CrudRepository` е единицата, за която ще се извлича информация от базата данни (в случая `Schedule`). Вторият параметър е типът на първичния ключ на единицата (в случая `Long`).

Сега може да се създаде обект от тип `ScheduleRepository` и чрез него да се извлича информация от базата данни – например методът `existsById()` ще върне дали съществува даден записан час (график) в базата данни, а `findAll()` ще върне всички записани часове от там. Не се налага разработчикът да настройва нищо друго, Spring Data JPA ще свърши работата вместо него.



Фиг.25. Клас диаграма на `ScheduleRepository`

Интересното в този интерфейс е, че има и 2 метода, които са ръчно декларирани – `findAllByPatient()` и `findAllOrderByDateAscTimeAsc()`, но никъде нямат имплементация.

Spring Data JPA автоматично имплементира custom методи за манипулиране на базата данни чрез специална конвенция за именуване на методи в интерфейс, който наследява от `CrudRepository` (или `JpaRepository`). Чрез тази конвенция могат да се създават методи, които сортират данните, търсят в тях по някой по-специфичен критерий и др. [14] Методът `findAllByPatient()` открива всички записани часове на даден пациент (не по `id`, а вместо това се подава самият `Patient` обект), а `findAllByOrderByDateAscTimeAsc()` връща всички записани часове, сортирани по дата и час във възходящ ред (този списък се използва от Thymeleaf, за да бъдат изведени първо най-старите записани часове в “schedule.html” страницата – това е обяснено по-надолу).

Пакет: `net.dvt32.DentistManager.service`

Файлове: **`MysqlBaseService.java`**, **`MysqlExportService.java`**, **`MysqlImportService.java`**

Описание:

Тези файлове са взети от open-source проекта [mysql-backup4j](#) [7] и са модифицирани, за да работят за Зъболекар 2.0. Чрез тези класове може да се импортира или експортира базата данни под формата на SQL файл. Оригиналният проект се разпространява и чрез Maven, но тази версия съдържа някои бъгове в свързването към база данни чрез JDBC адрес и затова файловете ръчно са добавени към Зъболекар 2.0. Промените по кода са обяснени в коментар в самите класове.

Макар импортирането/експортирането директно от SQL файл да представлява риск за сигурността, инструментът се грижи да не импортира данни, които не са били преди това експортирани с него. Работата с тези класове става от “archive.html” (обяснено надолу).

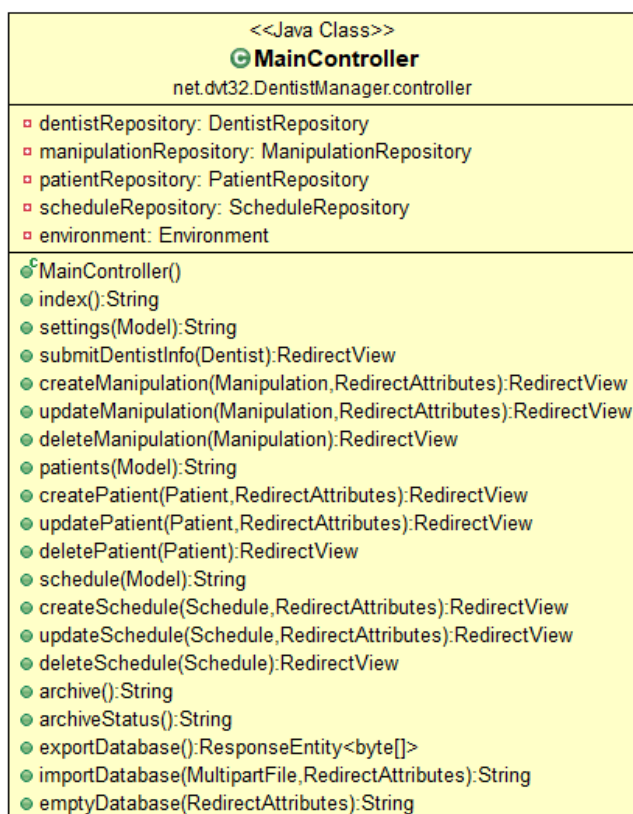
Пакет: `net.dvt32.DentistManager.controller`

Файл: **`MainController.java`**

Описание:

Този клас играе най-голяма роля в back-end функционалността на приложението. От него се осъществява най-съществената част – добавяне на информация в базата данни, изтриване на информация от там и обновяването ѝ.

Класът е аотиран с @Controller, което подсказва на Spring, че това е web контролер, тоест клас, който определя какво се случва, когато потребителят отвори някоя страница (например заглавната) или пък изпрати някаква информация през HTML форма.



Фиг.26. Клас диаграма на MainController

Класът има само 5 полета и те представляват хранилища, от които се извлича информация за единиците в базата данни. Полето „environment” се ползва за извличане на стойностите на настройките в *application.properties* файла. Полетата имат анотацията @Autowired, за да може Spring автоматично да ги инициализира (например хранилищата да могат да извличат и запазват данни, а пък „environment” да може да извлича данни от *application.properties*).

Методът ***index()*** има анотация `@RequestMapping("/")` и отговаря за това какво ще се случи, когато потребителят отвори заглавната страница на приложението (именно това означава частта „/” в анотацията – пътят към главната страница). Той връща String `“index”`, което означава, че Spring ще върне страницата `„index.html”`, която се намира в `src/main/resources/templates` (обяснено по-долу), но името на файла няма да се показва в URL адреса. В случая името на метода няма никакво значение – важен е адресът в анотацията.

Методът ***settings()*** има анотация `@RequestMapping("/settings")` и отговаря за това какво ще се случи, когато потребителят отвори страницата `settings` (тоест <адрес на приложението>/settings). Има и параметър от тип `Model` – в този обект се съхраняват данните, които да бъдат изпратени към `“settings”` страницата. В случая се извлича информацията за стоматолога от базата данни и информацията за всички манипулации. При зареждане на страницата, Thymeleaf използва подадената информация и я показва на потребителя (обяснено надолу). Методът връща String `“settings”`, което означава, че Spring ще върне страницата `„settings.html”`, която се намира в `src/main/resources/templates`, но името на файла няма да се показва в URL адреса.

Методът ***submitDentistInfo()*** има анотация `@PostMapping("/submitDentistInfo")` и отговаря за това какво ще се случи, когато потребителят натисне бутона „Запази“ на HTML формата с данните за стоматолога. Има параметър от тип `Dentist` с анотация `@ModelAttribute` – Spring успява да събере подадените данни от формата и да формира `Dentist` обект от тях (това важи и за методите, които манипулират пациенти, манипулации и записани часове). От там този обект може да бъде запазен директно в базата данни чрез съответното хранилище (в случая `dentistRepository`). Тъй като приложението поддържа само един стоматолог, `id`-то на този стоматолог винаги е 1 и ръчно се задава всеки път, за да не се добавя нов стоматолог към базата данни при натискане на бутона „Запази“ във формата, а вместо това да се обнови информацията за вече съществуващия. Методът връща `RedirectView` обект) към страницата `„settings.html”`, за да може след запазване на информацията потребителят да се върне на същата страница, но този път с обновени данни за стоматолога.

Методът ***createManipulation()*** има анотация `@PostMapping(value = "/submitManipulationInfo", params="createManipulation")` и отговаря за това какво ще се

случи, когато потребителят натисне бутона „Нова“ на HTML формата за създаване на манипулация в страницата „settings“. Тъй като всички бутони под формата препращат към „submitManipulationInfo“, трябва да стане ясно, че всеки бутон извършва различна дейност и затова чрез частта „params“ се задава името на метода, който да бъде изпълнен при натискане на „Нова“. Методът следи за невалидни данни и в такъв случай препраща към страница за грешка чрез своя RedirectAttributes параметър. Ако пък от формата е избрана някаква съществуваща манипулация, а след това потребителят е променил нещо по данните и накрая е натиснал „Нова“ вместо „Запази“, id-то на манипулацията временно получава стойност -1 и това позволява да се създаде нова манипулация, вместо да се обнови старата. Методът връща препращане (RedirectView обект) към страницата „settings.html“, за да може след запазване на информацията потребителят да се върне на същата страница, но този път с обновени данни.

Методите **updateManipulation()** и **deleteManipulation()** работят на същия принцип като createManipulation(). Разликата е, че в updateManipulation() няма нужда id-то на манипулацията да присвоява стойност -1, защото целта е просто да се обнови съществуваща манипулация (а това значи, че тя вече притежава валидно id) и че този метод се извиква при натискане на бутона „Запази“ под формата за манипулации в „settings“ страницата. deleteManipulation() изтрива манипулация, но само ако е избрана съществуваща такава от HTML формата, а не просто са попълнени някакви данни и след това е натиснат бутонът „Изтрий“ (в такъв случай просто се препраща към страницата „settings“ без да е извършено каквото и да било изтриване от базата данни).

Методът **patients()** има анотация @RequestMapping("/patients") и отговаря за това какво ще се случи, когато потребителят отвори страницата patients (тоест <адрес на приложението>/patients). Има и параметър от тип Model – в този обект се съхраняват данните, които да бъдат изпратени към „patients“ страницата. В случая се извлича информацията за всички пациенти в базата данни. Методът връща String „patients“, което означава, че Spring ще върне страницата „patients.html“, която се намира в src/main/resources/templates, но името на файла няма да се показва в URL адреса.

Методът **createPatient()** има анотация @PostMapping(value = "/submitPatientInfo", params="createPatient") и отговаря за това какво ще се случи, когато потребителят

натисне бутона „Нов“ на HTML формата за създаване на пациент в страницата „patients“. Тъй като всички бутони под формата препращат към „submitPatientInfo“, трябва да стане ясно, че всеки бутон извършва различна дейност и затова чрез частта „params“ се задава името на метода, който да бъде изпълнен при натискане на „Нов“. Методът следи за невалидни данни и в такъв случай препраща към страница за грешка чрез своя RedirectAttributes параметър. Ако пък от формата е избран някакъв съществуващ пациент, а след това потребителят е променил нещо по данните и накрая е натиснал „Нов“ вместо „Запази“, id-то на пациента временно получава стойност -1 и това позволява да се създаде нов пациент, вместо да се обнови старият. В същото време се създават и зъбите на пациента и за всеки от тях се посочва (чрез setter метода на Tooth setPatient()), че този зъб принадлежи на текущия пациент. Методът връща препращане (RedirectView обект) към страницата „patients.html“, за да може след запазване на информацията потребителят да се върне на същата страница, но този път с обновени данни.

Методите **updatePatient()** и **deletePatient()** работят на същия принцип като createPatient(). updatePatient() обновява данните за даден пациент, а в случай, че всъщност не е бил избран съществуващ пациент, а потребителят просто е въвел някакви данни и е натиснал бутона „Запази“ вместо „Нов“, то тогава просто се създава нов пациент. deletePatient() изтрива пациент (както и всички записани часове, които са свързани с него), но само ако е избран съществуващ такъв от HTML формата, а не просто са попълнени някакви данни и след това е натиснат бутонът „Изтрий“ (в такъв случай просто се препраща към страницата „patients“ без да е извършено каквото и да било изтриване от базата данни).

Методът **schedule()** има анотация @RequestMapping("/schedule") и отговаря за това какво ще се случи, когато потребителят отвори страницата schedule (тоест <адрес на приложението>/schedule). Има и параметър от тип Model – в този обект се съхраняват данните, които да бъдат изпратени към „schedule“ страницата. В случая от базата данни се извлича информацията за всички пациенти, всички манипулации и всички записани часове (графици), подредени по дата и час във възходящ ред. Методът връща String „schedule“, което означава, че Spring ще върне страницата „schedule.html“, която се намира в src/main/resources/templates, но името на файла няма да се показва в URL адреса.

Методът ***createSchedule()*** има анотация `@PostMapping(value = "/submitScheduleInfo", params="createSchedule")` и отговаря за това какво ще се случи, когато потребителят натисне бутона „Нов“ на HTML формата за създаване на график в страницата „schedule“. Тъй като всички бутони под формата препращат към „submitScheduleInfo“, трябва да стане ясно, че всеки бутон извършва различна дейност и затова чрез частта „params“ се задава името на метода, който да бъде изпълнен при натискане на „Нов“. Методът следи за невалидни данни и в такъв случай препраща към страница за грешка чрез своя `RedirectAttributes` параметър. Ако пък от формата е избран някакъв съществуващ график, а след това потребителят е променил нещо по данните и накрая е натиснал „Нов“ вместо „Запази“, id-то на графика временно получава стойност -1 и това позволява да се създаде нов график, вместо да се обнови старият. В същото време чрез setter методи се определят стойностите на всички атрибути на графика. Методът връща препращане (`RedirectView` обект) към страницата „schedule.html“, за да може след запазване на информацията потребителят да се върне на същата страница, но този път с обновени данни.

Методите ***updateSchedule()*** и ***deleteSchedule()*** работят на същия принцип като `createSchedule()`. `updateSchedule()` обновява данните за даден записан час, а в случай, че всъщност не е бил избран съществуващ график (записан час), а потребителят просто е въвел някакви данни и е натиснал бутона „Запази“ вместо „Нов“, то тогава просто се създава нов график. `deleteSchedule()` изтрива график, но само ако е избран съществуващ такъв от HTML формата, а не просто са попълнени някакви данни и след това е натиснат бутонът „Изтрий“ (в такъв случай просто се препраща към страницата „schedule“ без да е извършено каквото и да било изтриване от базата данни).

Методите ***archive()*** и ***archiveStatus()*** имат анотация `@RequestMapping` и отговарят за това какво ще се случи, когато потребителят отвори страниците `archive` и `archive-status` (тоест <адрес на приложението>/archive и <адрес на приложението>/archive-status). Първият метод връща String „archive“, което означава, че Spring ще върне страницата „archive.html“, която се намира в `src/main/resources/templates`, но името на файла няма да се показва в URL адреса (това де факто е категорията „Архивиране“ в системата). Вторият метод връща String, който препраща към страницата, която показва дали дадена операция по архивирането е била успешна или не (тоест дали импортирането или експортирането е било успешно), а ако не е била извършена такава операция, ще се появи страница за грешка.

Методът ***exportDatabase()*** има анотация `@RequestMapping("/exportDatabase")` и отговаря за това какво ще се случи, когато потребителят отвори страницата `exportDatabase` (тоест <адрес на приложението>/`exportDatabase`) или когато натисне бутона за експортиране в страницата `archive` (бутонът “Запази като SQL”). Този метод отговаря за експортирането на данните от базата данни и използва класовете от пакета `net.dvt32.DentistManager.service`. Първо прочита JDBC адреса и потребителското име / паролата за базата данни от `application.properties` файла (за да може да се установи връзка с нея), а след това генерира SQL от таблиците вътре. Методът връща генерирания SQL файл като `ResponseEntity` обект.

Методът ***importDatabase()*** има анотация `@PostMapping("/importDatabase")` и отговаря за това какво ще се случи, когато потребителят се опита да възстанови база данни от избран от него SQL файл чрез бутона „Възстанови от SQL” в страницата „`archive`”. Методът следи за невалидни данни и в такъв случай препраща към страница за грешка чрез своя `RedirectAttributes` параметър, а при успех препраща към страница, която съобщава, че импортирането на данните е било успешно (и в двата случая се препраща към страницата `archive-status`, но съдържанието в нея е различно и Thymeleaf разбира кое съобщение да покаже според атрибутите изпратени от метода). В този метод се следи дали подаденият файл е валиден SQL файл, който е бил създаден през класовете в пакета `net.dvt32.DentistManager.service`. Ако това е така, кодът бива прочетен от файла и след това се вмъкват данните от него (като предварително се изпразват таблиците в базата данни).

Методът ***emptyDatabase()*** работи на същия принцип като `importDatabase()` и отговаря за това какво ще се случи, когато потребителят натисне бутона „Изпразни” в страницата „`archive`”, но разликата е, че тук не се подава файл, а просто подаденият SQL код е празен `String` (по-точно стойността му е един интервал, защото класовете в пакета `net.dvt32.DentistManager.service` не приемат като аргумент изцяло празен `String`) и в резултат на това единственото нещо, което се случва, е, че таблиците в базата данни биват изпразнени.

4.2. Front-end функционалност (клиентска част)

Файл: **`static/css/main.css`**

Описание: CSS файл, в който е посочен цветът на фона на всяка страница в приложението. Във файла липсват други CSS правила, защото преобладаващата част от дизайна на приложението се реализира от Bootstrap.

Файл: **static/js/pdfmake.min.js**

Описание: JavaScript файл, който е част от pdfmake библиотеката, която се използва за генериране на PDF отчет за състоянието на зъбите на даден пациент.

Файл: **static/js/vfs_fonts.js**

Описание: JavaScript файл, който е част от pdfmake библиотеката. В този файл се определят шрифтовете, които се използват в генерирания PDF файл.

Файл: **templates/error/404.html**

Описание:

Страница за грешка, която се зарежда, когато потребителят се опитва да отвори страница, която не съществува. Spring автоматично открива файлове в папката error с име кода на грешката (например 404.html, 500.html и т.н.) и ги показва, когато възникне съответната грешка.

Както и във всички други страници в приложението, има 2 meta тага в <head> частта на страницата. Първият определя какъв тип символно кодиране се използва (UTF-8 стандартът е най-подходящ за кирилица), а вторият прави страницата „responsive“, тоест я прави достъпна и от мобилни устройства (елементите запазват своята форма, но се подреждат по начин, който е по-подходящ за екрана на мобилното устройство).

Отделно са вмъкнати Bootstrap и jQuery. Класовете в Bootstrap формират дизайна на страницата (както и на всички други) и затова менютата имат много класове като „navbar-item“ и „bg-secondary“, които определят как са разположени елементите, в какви цветове са и още. [15]

Използвана е и библиотеката Emoji CSS за лесно вмъкване на емоji изображение в съобщението за грешка.

Файл: **templates/archive-status.html**

Описание:

Страница, която показва статуса на някоя извършена операция по архивирането на базата данни (импортиране или експортиране на данните вътре). Страницата използва Thymeleaf за избор на съобщението, което да се покаже пред потребителя. Според статус атрибута, който е бил изпратен от метода за архивиране на базата данни в Spring web контролера (класът MainController), Thymeleaf показва различен текст (или за успешна операция, или за провалена такава). Thymeleaf се вмъква в страницата чрез реда „<html xmlns:th="http://www.thymeleaf.org">“, а частта „th:if="{archiveStatusMessage.equals('...')}" в долните div тагове казва „ако условието е изпълнено, покажи съдържанието в този div таг“.

Файл: **templates/archive.html**

Описание:

Страница, която отговаря за менюто „Архивиране“ в системата. От тук може да се възстанови базата данни от файл, да се експортира в SQL формат или да се изпразни.

За всяко действие има отделна форма (form таг), която отговаря за изпълнението му. Всяка форма има атрибут *action* с името на метода, който се изпълнява от Java класа web контролер (MainController) през Spring, когато потребителят натисне бутона за изпращане. Тъй като импортирането и изпразването са необратими операции, преди да бъде изпълнено действието се задейства JavaScript confirm() диалог, който подканва потребителя първо да потвърди, че иска да извърши това действие (изпълнява се кодът в *onsubmit* атрибута на form таговете).

Формите и бутоните са стилизирани с *col* класовете на Bootstrap (чрез тях се указва размерът на полето, което заемат), а пък външното отстояние е настроено чрез класовете *mt* (margin-top) и *mb* (margin-bottom).

Формата за експортиране е с *get* стойност на *method* атрибута, защото се извлича информация от базата данни, а за другите две операции (импортиране и изпразване) е по-подходяща *post* стойност, защото се изпраща информация към сървъра.

Файл: **templates/error.html**

Описание:

Обща страница за грешки – например при изпращане на някаква информация (примерно при създаването на пациент или график) или при някаква вътрешна сървърна грешка. Spring автоматично открива файла „error.html” и го зарежда при възникването на каквато и да е грешка (освен ако няма по-специфичен файл като 404.html в папка error).

Thymeleaf избира подходящо съобщение, което да покаже на потребителя на база на това дали има изпратени някакви атрибути от web контролера. Ако няма никакви изпратени атрибути – съобщението за грешка е общо, а в противен случай по-конкретно (например грешка за повтарящо се ЕГН, повтарящо се име на манипулация и други).

Файл: **templates/index.html**

Описание: главна страница на приложението – използва класове на Bootstrap за стилизиране (*card*, *col*, *mt*, *mb* и други). Класът „card” се използва за създаване на гъвкав HTML контейнер за съдържанието. [16]

Файл: **templates/patients.html**

Описание:

Страница, която отговаря за менюто „Пациенти” в системата. От тук може да се създаде пациент, да се обнови информацията за него или да бъде изтрита. Може и да се принтира PDF отчет за състоянието на зъбите му (отчетът включва и попълнените за него данни). В <head> частта на страницата е вмъкната JavaScript библиотеката pdfmake, която е нужна за генериране на PDF отчета. Както и при другите страници се използва класът “navbar” на Bootstrap, чрез който се реализира главното меню най-отгоре (хедърът, който съдържа линкове към различните страници в системата). [17]

Списъкът от пациентите се показва благодарение на Thymeleaf, който използва цикъл, за да покаже данните на всички пациенти, които са били изпратени от метод в web контролера (*patients()* методът в MainController класа). Също така чрез Thymeleaf се

настройва ширината на `select` тага, който съдържа имената на пациентите (ако няма пациенти, тагът има нормална ширина, а в противен случай ширината се настройва, така че да може да се види цялото име на пациента – това става чрез атрибута `“th:style”`). Когато е избран пациент, данните му автоматично биват попълнени от jQuery през метода *fillPatientInfo()* (затова атрибутът `onchange` има стойност името на този метод).

След това идва формата за данните на пациента. Тя има атрибут *onsubmit* със стойност JavaScript код, който се изпълнява преди информацията да бъде изпратена на сървъра (кодът е обяснен по-надолу). Всеки пациент в началото получава стойност за своето `id` -1 (има невидимо поле, което съдържа тази стойност). По този начин при изпращане на информацията на сървъра, Hibernate генерира уникален номер за пациента и след това го добавя в базата данни. Ако е избран съществуващ пациент, Hibernate не променя `id`-то и просто запазва обновената информация за пациента без да създава нов такъв (освен ако не е бил натиснат бутонът „Нов“).

Надолу следват самите полета, които съдържат данните за пациента. Всяко поле има атрибут `„name”` и чрез него Spring свързва всяко поле с променливата в Java класа, която отговаря на това поле. Полетата имат и `id` атрибут, за да може по-лесно да бъдат манипулирани чрез jQuery. Пространството, което елементите (полетата) заемат се определя автоматично чрез `col`, `form-group` [18] и `form-check` класовете на Bootstrap.

Преди таблицата за състоянието на зъбите има 2 бутона – единият за генериране на PDF отчет (неактивен, ако не е избран съществуващ пациент) и другият за показване на „легенда” (различни цветове съответстват на различно състояние на зъб). Първият изпълнява функцията *generatePatientPdfReport()* при натискане (обяснена надолу), а вторият показва падаща таблица с различните цветове, които съответстват на зъбни състояния като кариес, корен и така нататък (това се осъществява чрез JavaScript plugin-а „Collapse” на Bootstrap). [19] Отдолу е реализирана тази таблица и е стилизирана с различни Bootstrap класове.

Следва таблица, чрез която се определя състоянието на всеки зъб. По подразбиране състоянието на всеки зъб е „Здрав“, но това може да се промени чрез модалния прозорец, който излиза, когато потребителят натисне върху полето на някой от зъбите. Това държане на таблицата се осъществява чрез JavaScript plugin-а „Modal” на

Bootstrap. Всеки елемент има `data-target` атрибут със стойност `id`-то на модалния прозорец, който трябва да бъде показан. Елементите имат и атрибут `id`, който позволява по-лесна манипулация с jQuery по-надолу. Важно е да се отбележи, че номерата на зъбите в двата атрибута не са еднакви, защото в `data-target` атрибута е показан номерът на зъба според таблицата, а в `id` атрибута е показан индексът на зъба в масива, в който се съхранява информацията за зъбите на пациента.

Кодът на самите модални прозорци е под този на таблицата за състоянието на зъбите. Реализацията следва примера за създаване на модални прозорци в документацията на Bootstrap. [20] Във всеки такъв прозорец се съдържа и списък с извършените манипулации върху конкретния зъб (елементите с `id` `"teeth[индекс-на-зъба].appliedManipulations"`), а тази информация се попълва от JavaScript код, който е по-надолу.

Трите бутона най-отдолу изпращат информацията от формата, но всеки има различно предназначение. Според натиснатия бутон се извиква различен метод в Spring контролера (името на метода отговаря на стойността на `name` атрибута на бутона). При натискане на някой от бутоните се изпълнява JavaScript функцията `submitPatientInfoOnSubmit()` и ако натисният бутон е „Изтрий“ се появява `confirm()` диалог за потвърждение (тъй като изтриването е необратима операция и има вероятност потребителят да иска да се откаже от решението си).

Следващият скрипт декларира някои глобални променливи и „инициализира“ стойностите в `select` таговете в страницата. Масивът `selectItems` включва всичките валидни състояния на един зъб и те биват добавени като опции в `select` тага на всеки модален прозорец в страницата. След това е декларирана глобална таблица (Map обект), който съдържа HEX кодовете на цветовете (`toothCellHexColors`), които съответстват на всяко зъбно състояние. Чрез тази информация jQuery настройва стойностите на `name` атрибутите на елементите, които представят състоянието на зъбите, а в същото време настройва и цвета на клетките в таблицата за състоянието на зъбите (при зареждане на страницата всички клетки са в бяло).

Функцията `fillPatientInfo()` използва възможностите на Thymeleaf и т.нар. „script inlining“. [21] Тя прочита подадения списък с пациенти от Spring контролера, намира текущия избран пациент и чрез jQuery информацията за този пациент бива попълнена във

формата, която вижда потребителят. Това включва промяна на стойностите в текстовите полета, промяна на цветовете в таблицата за състояние на зъбите на пациента и всичко останало.

Последната функция *generatePdfPatientReport()* генерира PDF файл отчет за пациента на база на наличните данни за него. Отново се извлича информацията чрез Thymeleaf и със специално структуриран масив (*docDefinition*) pdfmake генерира PDF файла според своя специален синтаксис. [22] Тук влиза в употреба и глобалния Map обект с цветовете стойности, за да може състоянието на зъбите да бъде точно илюстрирано и в изходния файл. Името на файла по подразбиране е „patient-report.pdf”, но потребителят може да го промени преди запазването му.

Файл: **templates/schedule.html**

Описание:

Страница, която отговаря за менюто „График“ в системата. От тук може да се създаде график (да се запише час за пациент), да се обнови информацията за него или да бъде изтрита. Начинът на работа на тази страница доста наподобява този на “patients.html” с някои малки разлики.

Дизайнът, формите и JavaScript кодът за попълване и манипулиране на данните са структурирани по същия начин както в “patients.html”, но разликата е, че тук на всеки записан час съответства по някой пациент от базата данни и затова 2 функции попълват информацията във формата (*fillPatientInfo()* и *fillScheduleInfo()*). Отделно се извлича и информацията за наличните манипулации, които могат да бъдат поставени на зъбите, а самото поставяне става чрез модалния прозорец, който се появява при натискане на левия бутон на мишката върху клетката за състоянието на даден зъб.

Във функцията *fillScheduleInfo()* се изчислява дължимата от пациента сума според цената на поставените манипулации (с включено ДДС). При промяна на избрания график се скриват предишните заредени стойности в полетата на формата и се попълват нови (тези на новия избран график) чрез jQuery.

Файл: **templates/settings.html**

Описание:

Страница, която отговаря за менюто „Настройки“ в системата. От тук могат да се попълнят личните данни на стоматолога и да се създават, обновяват или изтриват манипулации, които ще бъдат поставени върху зъби на пациенти.

Начинът на работа на тази страница е подобен на този на „patients.html“ и „schedule.html“, но тук разликата е, че данните за стоматолога и тези за манипулациите са разделени с табове (тоест подменюта), които се реализирани чрез JavaScript plugin на Bootstrap. [23]

Данните за стоматолога не биват попълнени от JavaScript функция, а директно чрез Thymeleaf (чрез th:value атрибут на всяко негово поле), защото стоматологът е само един и няма нужда да се създава хранилище, което да се изпраща през Spring контролера (изпраща се само един Dentist обект). Тъй като се използват радио бутони за настройване на пола на стоматолога, а тези бутони са стилизирани с Bootstrap, се използва и атрибутът th:classappend, за да се добави класът active, който променя това как изглежда бутонът (тоест бутонът, който отговаря на пола на стоматолога, да изпъква, а другият не).

Функцията *fillManipulationInfo()* извлича данните за манипулациите от базата данни и от там попълва формата с данните на избраната от потребителя манипулация. Ако потребителят натисне бутона „Изтриване“ се показва confirm() диалог, който го пита дали е сигурен, че иска да извърши това (тъй като изтриването е необратима операция). Това става благодарение на функцията *submitManipulationInfoOnSubmit()*, която се задейства при изпращане на формата (но ефект от нея има само при натискане на бутона „Изтриване“).

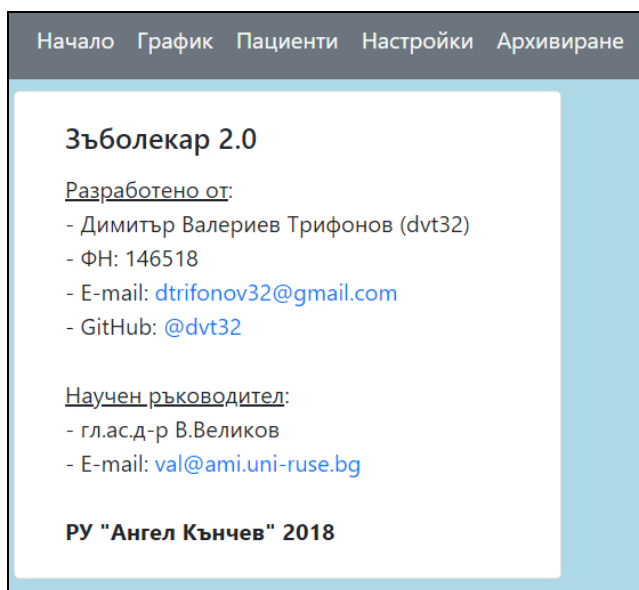
5. Ръководство за работа със системата

5.1. Системни изисквания

Зъболекар 2.0 изисква само достъп до интернет и модерен уеб браузър като Google Chrome (препоръчително), Mozilla Firefox или Opera. Необходимо е и JavaScript да е позволен в браузъра.

5.2. Начин на работа (външна структура)

Системата има 4 категории: **График**, **Пациенти**, **Настройки** и **Архивиране** (това вижда потребителят). Началната страница просто дава малко информация за разработчика на приложението.



Фиг. 27. Начална страница на приложението

5.2.1. График

От тази страница потребителят може да управлява графика си – да види списък със записаните часове, да създаде нов такъв или да промени данните по него (може и да го изтрие). За всеки записан час се следи какви манипулации са били извършвани и дали дължимата сума е била изплатена от пациента. Часовете са подредени от най-стария към най-скорошния.

Начало График Пациенти Настройки Архивиране

График подреден по дата и час

(2018-10-01) @ 13:00 - Мария Иванова
(2018-10-01) @ 14:00 - Петър Петров
(2018-10-31) @ 14:00 - Димитър Димитров

Извършени услуги и цени:

12.00 лв. - Поставяне на коронка
2.40 лв. - Поставяне на лекарство

☒ Платено (Цена: 14.40 лв.)

Информация за графика

Дата *:

01.10.2018

* Задължително поле.

Избран час *:

13:00

* Комбинацията дата/час трябва да е уникална.
Задължително поле.

Фиг.28. Избиране на запазен час и настройване на дата/час за него

Всъщност не е необходимо потребителят ръчно да въвежда датата и часа – може да ползва вградения picker в брауъра (при всеки брауър той изглежда различно - долните изображения са от Google Chrome).

Дата *:

30.08.2018

August 2018

Sun	Mon	Tue	Wed	Thu	Fri	Sat
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1

Избран час *:

15:54

Фиг.29. Picker в Google Chrome за дата и час

Когато потребителят избере съществуващ час, данните за него автоматично се попълват в долната форма.

Също така от тук могат да се манипулират и състоянията на зъбите на даден пациент. От тук се поставят и манипулации върху тези зъби (чрез модален прозорец, който се

появява при натискане на ляв бутон на мишката върху клетка от таблицата за зъбите на пациента).

Пациент *:

Мария Иванова (9509101444)
Димитър Димитров (8909101444)
Петър Петров (9509101312)

* Задължително поле.

Статус на зъбите:

18	17	16	15	14	13	12	11	21	22	23	24	25	26	27	28
48	47	46	45	44	43	42	41	31	32	33	34	35	36	37	38

Забележка:

Запис Нов Изтриване

Зъб 13

Текущо състояние:
Обтурация

Извършвани манипулации:
(2018-10-01) @ 13:00 - Поставяне на коронка

Текуща манипулация:
Поставяне на коронка

Затвори

Фиг.30. Меню за поставяне на манипулация върху зъб на пациент

При натискане на бутона „Изтриване“ се показва диалог, който пита потребителя дали е сигурен, че иска да извърши тази необратима операция, а ако той натисне Cancel, формата не се изпраща.

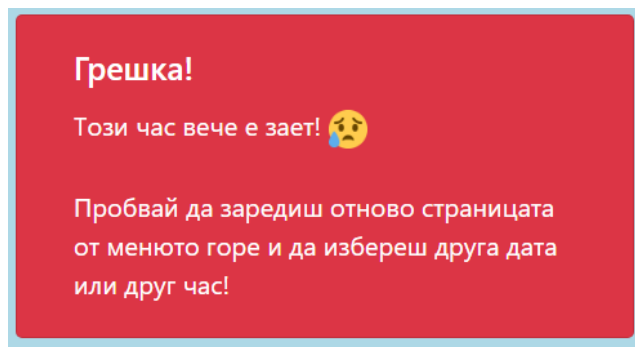
localhost:8080 says

Сигурен ли си, че искаш да изтриеш данните за този записан час?

OK Cancel

Фиг.31. Диалог за потвърждение при изтриване на час

При въвеждане на невалидна комбинация от дата и час (тоест вече има записан час по това време) се появява страница за грешка.



Фиг.32. Съобщение за грешка при зает час

Бутонът „Запис“ се използва за обновяване на информацията на даден час, а „Нов“ за създаване на нов такъв в базата данни. Ако преди това не е избран съществуващ час от списъка със записани часове, „Запис“ прави същото като „Нов“.

5.2.2. Пациенти

От тази страница потребителят може да управлява пациентите в базата данни – да добавя информация за нов пациент, да променя данните на съществуващ такъв или изцяло да го изтрие.

Начало График Пациенти Настройки Архивиране

Списък с пациенти

- Мария Иванова - 9509101444
- Димитър Димитров - 8909101444
- Петър Петров - 9509101312

Данни за пациента

Име *:
Мария Иванова
* Задължително поле.

Телефон:
882261071

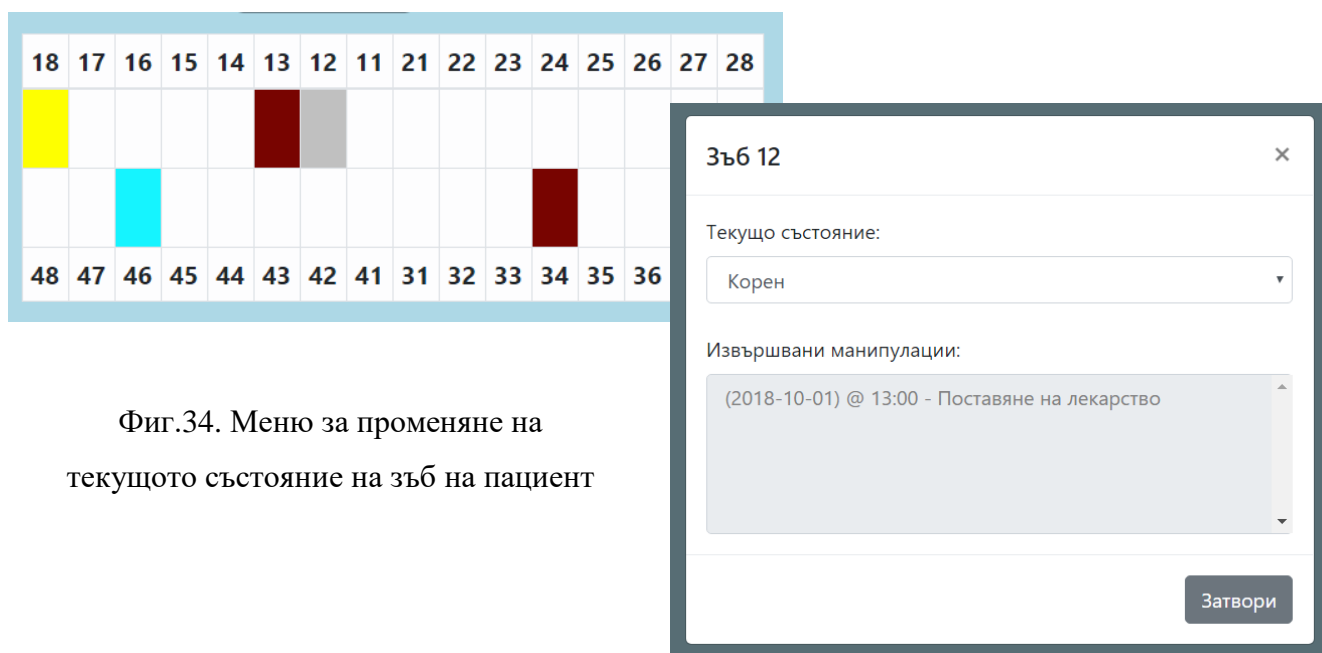
ЕГН *:
9509101444
* Номерът трябва да е уникален.
Задължително поле.

Пол:
Мъж Жена

Фиг.33. Избиране на пациент
настройване на данните за него

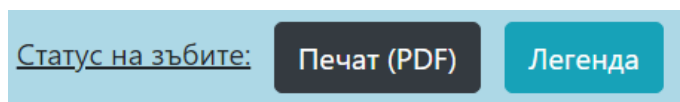
При избиране на съществуващ пациент от списъка, формата с данните за него автоматично се попълва.

Състоянието на зъбите на пациента се променя точно както в „График“ (чрез модален прозорец, който се зарежда при натискане на ляв бутон на мишката върху съответстващата клетка на зъба в таблицата за състояние на зъбите), но тук няма възможност да се поставят манипулации, а само се вижда списък с вече приложени манипулации върху дадения зъб.















Фиг.34. Меню за променяне на текущото състояние на зъб на пациент

Над таблицата има 2 бутона: за генериране на PDF отчет за пациента (бутонът е неактивен, ако не е избран съществуващ пациент) и за показване на легенда, която показва в падаща таблица кой цвят на кое зъбно състояние съответства.

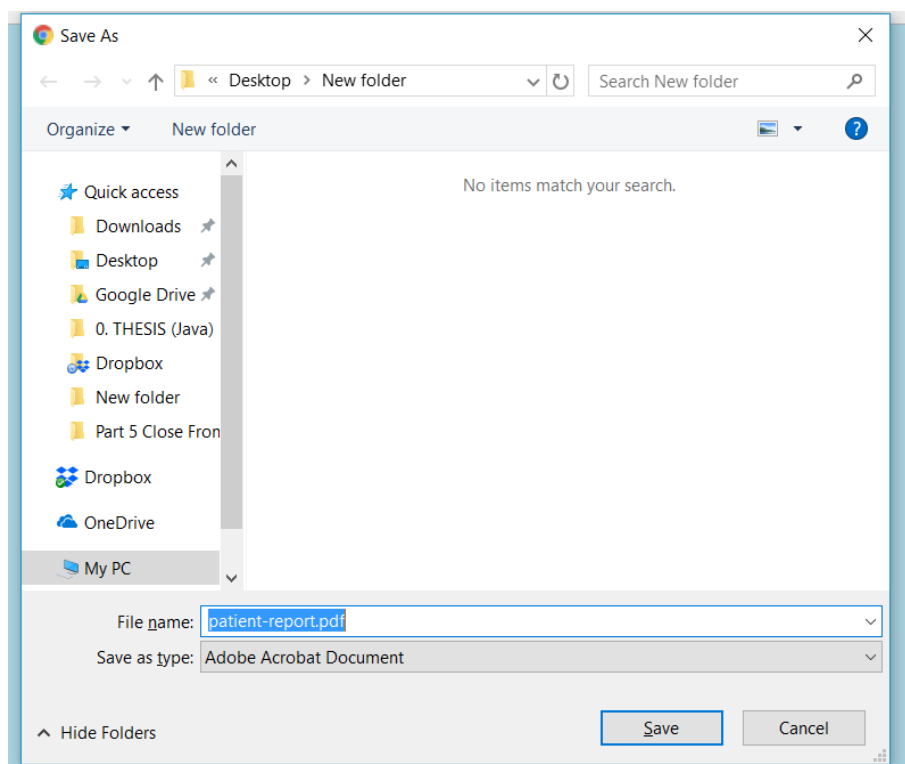


Фиг.35. Бутони за генериране на PDF отчет и легенда на зъбните състояния

	Кариес		Корен		Коронка		Пародонтоза
	Пулпит		Обтурация		Изкуствен		Периодонтит
	Гангрена		Липсващ		Фрактура		Пломба

Фиг.36. Легенда на зъбните състояния

При натискане на бутона “Печат” (PDF) се отваря меню от операционната система, което пита потребителя къде иска да запази отчета.



Фиг.37. Меню за запазване на PDF отчета под Windows 10

Самото съдържание на файла изглежда така:

Име на пациента: Мария Иванова
 Телефон: 882261071
 ЕГН: 9509101444
 Пол: жена
 Адрес:
 Кръвна група:
 Резус фактор:
 Алергии:
 Сърд.недостатъчност: не
 Диабет: не
 ХИВ: не
 Забележка:

Статус на зъбите:

18	17	16	15	14	13	12	11	21	22	23	24	25	26	27	28
Yellow					Dark Red	Grey									
		Cyan									Dark Red				
48	47	46	45	44	43	42	41	31	32	33	34	35	36	37	38

Зъб 18: Кариес
 Зъб 17: Здрав
 Зъб 16: Здрав
 Зъб 15: Здрав
 Зъб 14: Здрав
 Зъб 13: Обтурация
 Зъб 12: Корен
 Зъб 11: Здрав
 Зъб 21: Здрав
 Зъб 22: Здрав
 Зъб 23: Здрав
 Зъб 24: Здрав
 Зъб 25: Здрав
 Зъб 26: Здрав
 Зъб 27: Здрав
 Зъб 28: Здрав

Зъб 48: Здрав
 Зъб 47: Здрав
 Зъб 46: Коронка
 Зъб 45: Здрав
 Зъб 44: Здрав
 Зъб 43: Здрав
 Зъб 42: Здрав
 Зъб 41: Здрав
 Зъб 31: Здрав
 Зъб 32: Здрав
 Зъб 33: Здрав
 Зъб 34: Обтурация
 Зъб 35: Здрав
 Зъб 36: Здрав
 Зъб 37: Здрав
 Зъб 38: Здрав

Фиг.38. Съдържание на PDF отчета за пациента

Бутоните най-отдолу служат за изпращане на формата. „Запази“ обновява данните за пациента, „Нов“ създава нов такъв в базата данни, а „Изтриване“ изтрива избрания пациент (преди това се появява съобщение за потвърждение). В случай, че не е избран пациент, то „Нов“ и „Запази“ правят едно и също.

Запази Нов Изтриване

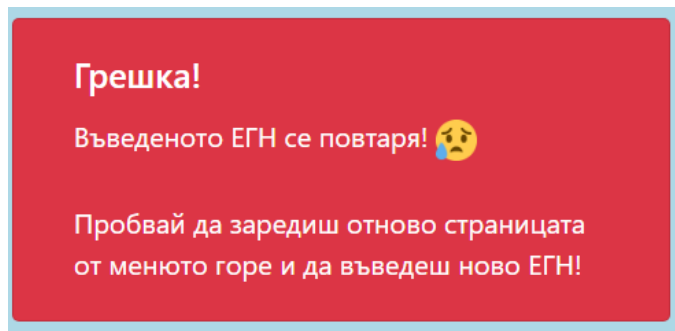
localhost:8080 says

Сигурен ли си, че искаш да изтриеш данните за пациента?

OK Cancel

Фиг.39 и 40. Бутони за изпращане на формата (ляво) и съобщение за потвърждение на изтриване на пациент от базата данни (дясно).

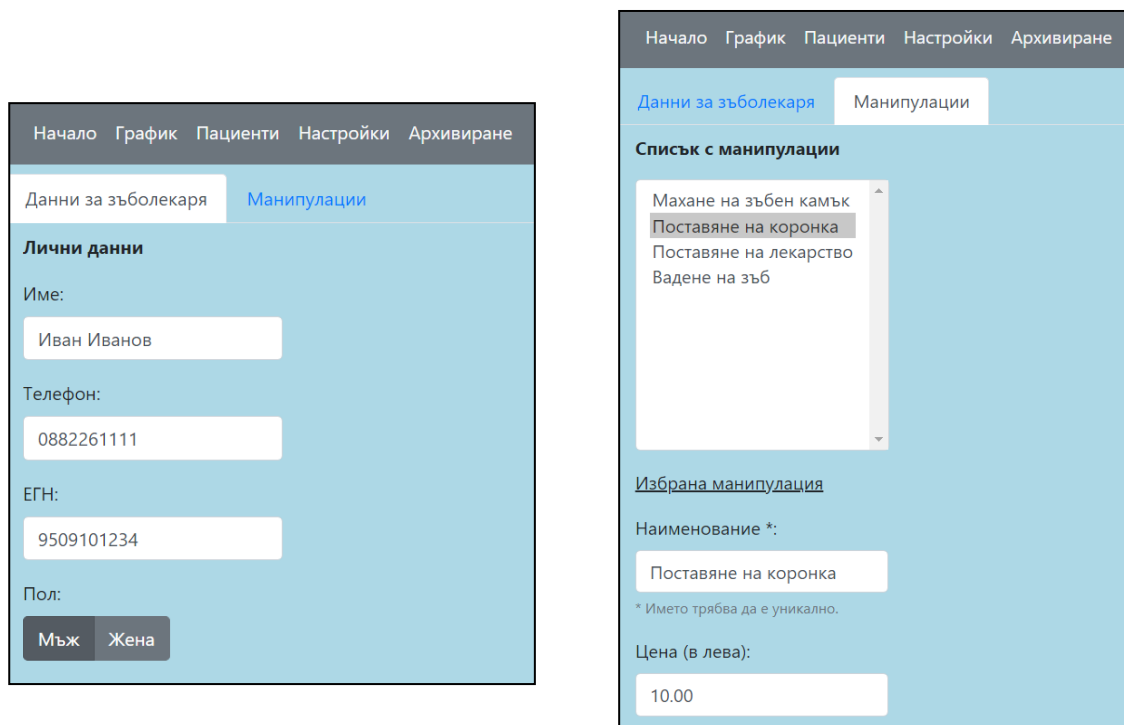
При въвеждане на повтарящо се ЕГН се появява страница за грешка след изпращане на формата.



Фиг.41. Съобщение за грешка при повтарящо се ЕГН

5.2.3. Настройки

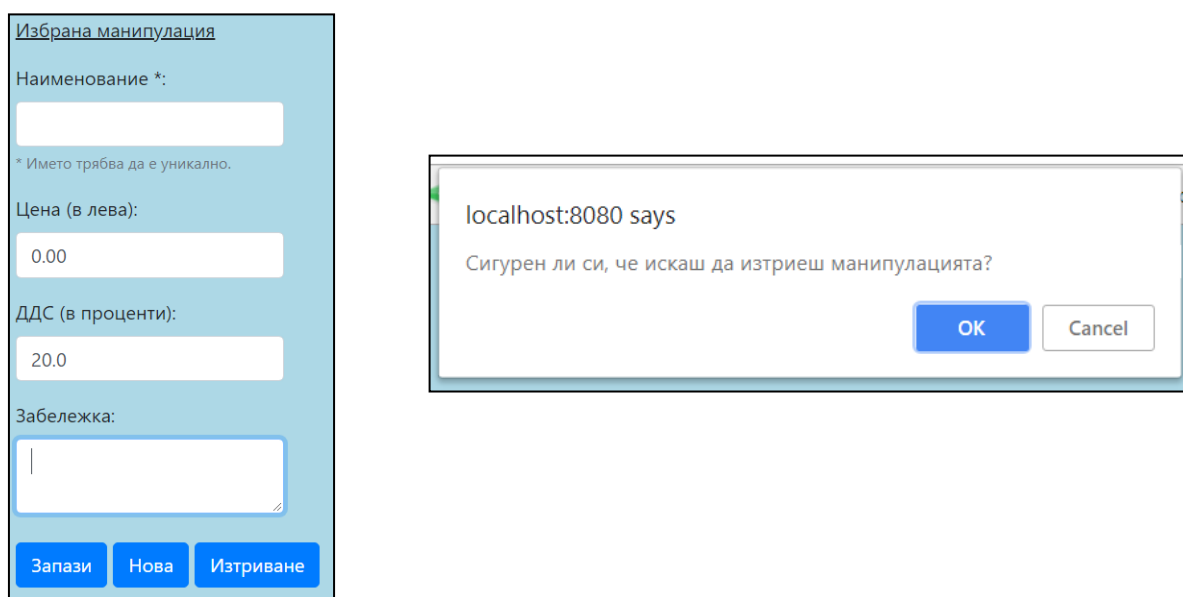
От тази страница могат да се попълнят данните на стоматолога и да се създават, обновяват или изтриват манипулации от базата данни. Данните на стоматолога и тези за манипулациите са разделени чрез табове (превключването става с натискане на ляв бутон на мишката върху таба, който потребителят иска да види).



Фиг.42. Таб с данни за зъболекаря (ляво) и таб с данни за манипулациите (дясно)

Чрез бутона „Запази“ в таба на стоматолога се запамятват данните за него и при следващото зареждане на страницата те ще бъдат автоматично попълнени във формата.

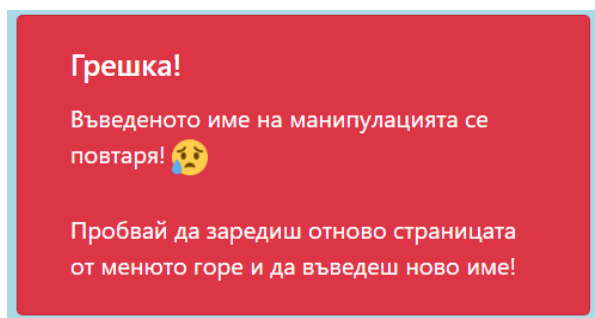
Бутоните най-отдолу в таба за манипулациите отговарят за изпращането на формата с данните за манипулациите. При избиране на съществуваща манипулация данните във формата се попълват автоматично. При натискане на бутона „Изтриване“ се появява съобщение за потвърждение преди да бъде изпратена информацията към сървъра. Ако не е избрана съществуваща манипулация предварително, „Нова“ и „Запази“ правят едно и също нещо (добавят нова манипулация в базата данни).



The image shows two parts of a web application. On the left is a form titled "Избрана манипулация" (Selected Manipulation). It contains several input fields: "Наименование *" (Name *) with a placeholder, a note "* Името трябва да е уникално." (The name must be unique.), "Цена (в лева):" (Price (in leva):) with the value 0.00, "ДДС (в проценти):" (DDS (in percent):) with the value 20.0, and a "Забележка:" (Note:) field. At the bottom are three buttons: "Запази" (Save), "Нова" (New), and "Изтриване" (Delete). On the right is a confirmation dialog box with the text "localhost:8080 says" and "Сигурен ли си, че искаш да изтриеш манипулацията?" (Are you sure you want to delete the manipulation?). It has "OK" and "Cancel" buttons.

Фиг.43. Форма за попълване на данни за манипулация (ляво) и съобщение за потвърждение на изтриване на манипулация (дясно)

При изпращане на манипулация с повтарящо се име се зарежда страница за грешка.

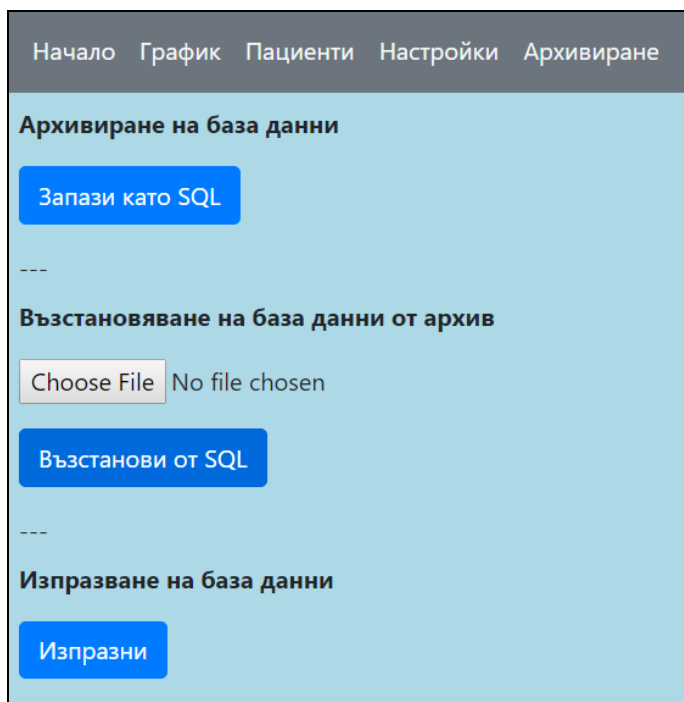


The image shows a red error message box with the text "Грешка!" (Error!) in bold. Below it, it says "Въведеното име на манипулацията се повтаря!" (The entered name of the manipulation is repeated!) followed by a sad face emoji. At the bottom, it says "Пробвай да заредиш отново страницата от менюто горе и да въведеш ново име!" (Try to reload the page from the menu above and enter a new name!).

Фиг.44. Съобщение за грешка при повтарящо се име на манипулация

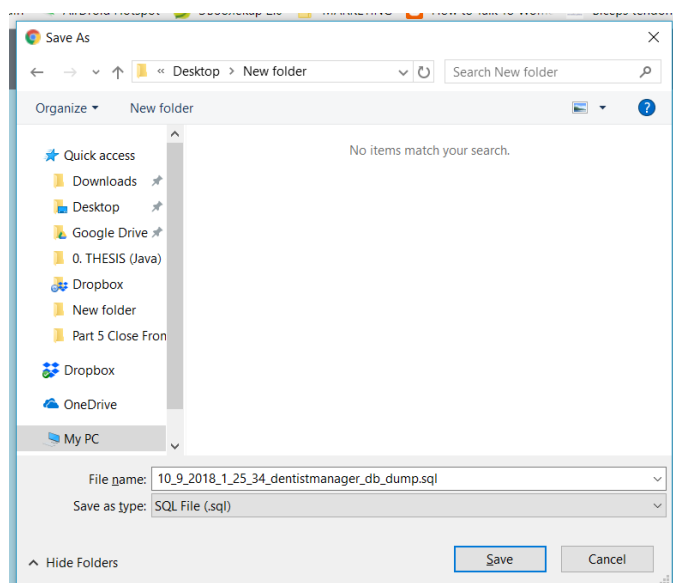
5.2.4. Архивиране

От тази страница потребителят може да експортира информацията от базата данни във SQL файл, да изпразни базата данни или да я възстанови от SQL файл, който е бил генериран от тук.



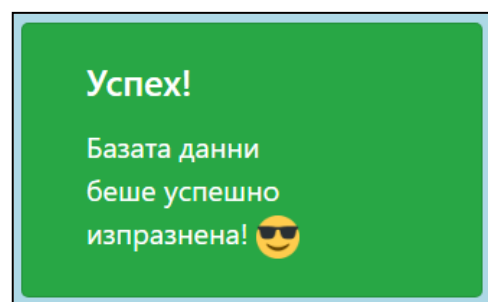
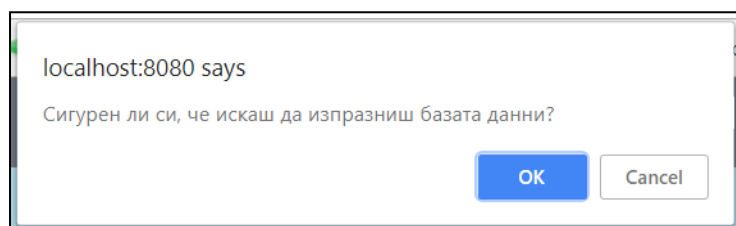
Фиг.45. Страница „Архивиране“ в системата

При натискане на бутона „Запази като SQL“ се отваря диалог от операционната система, който пита къде да бъде съхранен файлът.



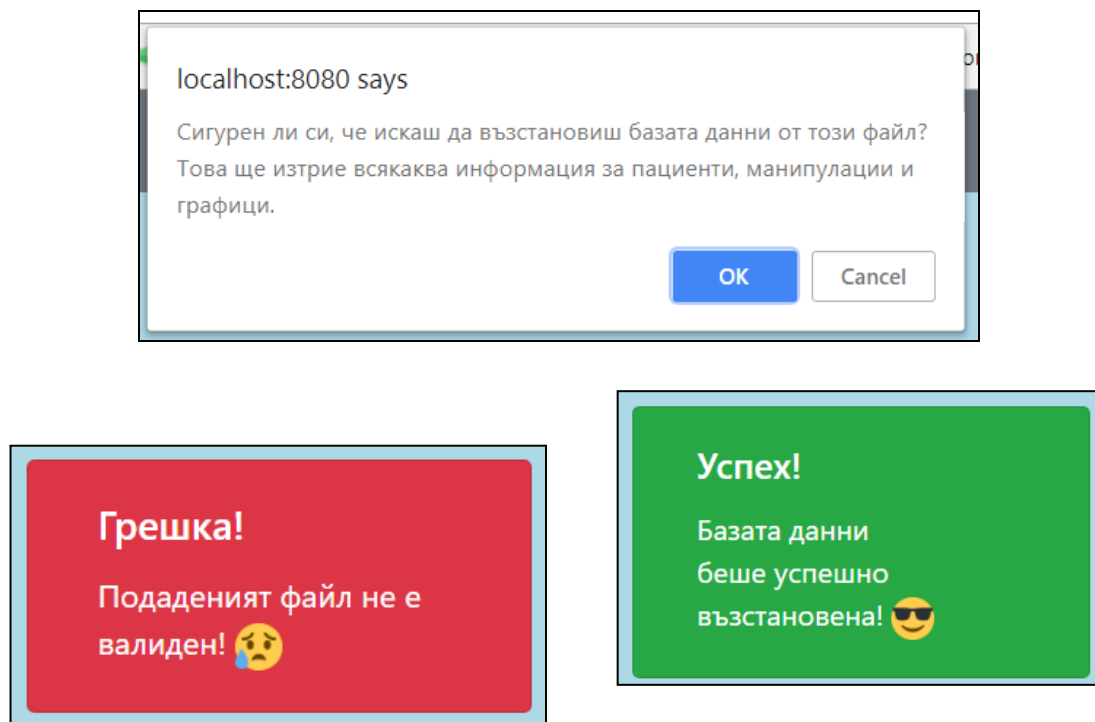
Фиг. 46. Диалог за запазване на базата данни като SQL файл

При натискане на бутона „Изпразни“ първо се появява прозорец за потвърждение (тъй като това е необратима операция и заличава цялата информация в базата данни), а в случай че потребителят натисне „ОК“, се зарежда страница, която съобщава, че изпразването е било успешно.



Фиг.47 и 48. Прозорец за потвърждение на изпразването на базата данни (ляво) и съобщение за успешно изпразнена база данни (дясно)

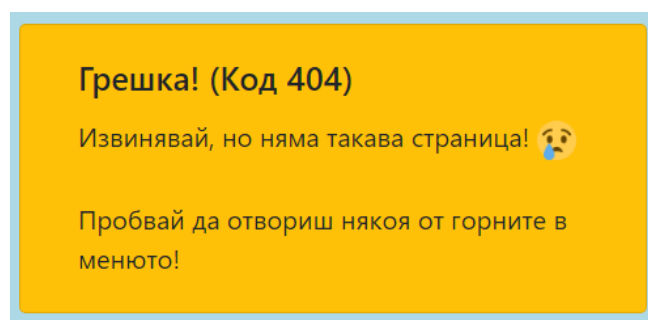
За да се възстанови базата данни от файл, първо трябва да се избере такъв през file picker-а над бутона „Възстанови от SQL“. Всеки браузър показва този file picker по различен начин. [24] Чрез този file picker се отваря меню от операционната система, което позволява на потребителя да избере SQL файл (по подразбиране се показват файлове само с това разширение, но това може да бъде и променено от потребителя). След избиране на файла натискането на бутона „Възстанови от SQL“ води или до страница със съобщение за успех (ако файлът е валиден), или до страница със съобщение за неуспех (ако файлът не е валиден), но преди това се показва прозорец за потвърждение на извършването на операцията.



Фиг.49 и 50. Прозорец за потвърждение на възстановяването на базата данни (горе) и съобщение за грешка/успех при възстановяването (долу)

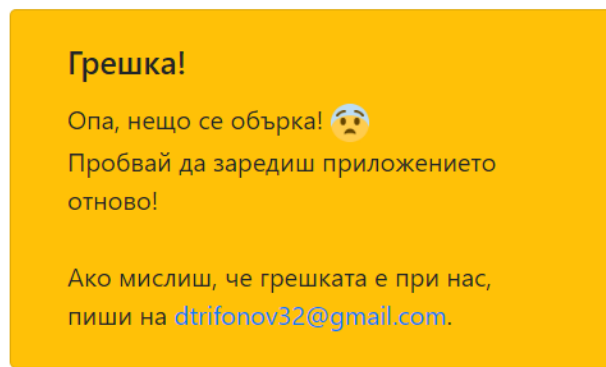
5.2.5. Страници за грешки

При отваряне на несъществуваща страница (в рамките на приложението) се зарежда страница за грешка (код 404).



Фиг.51. Съобщение за липсваща страница (грешка с код 404)

За всякакви други грешки се появява обща страница, която подканва потребителя да сподели какво е довело до тази грешка с разработчика на приложението.



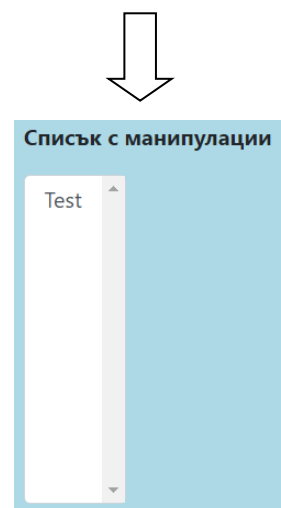
Фиг.52. Съобщение за обща/неизвестна грешка

5.3. Тестване на системата

Тук ще бъде илюстриран възможно най-общият случай на употреба (use case) на приложението – създаване на манипулация, добавяне на един пациент в базата данни и след това записване на час за него и поставяне на манипулацията върху един от зъбите му.

В началото базата данни е празна и първо създаваме манипулацията:

=> Манипулацията е успешно добавена в базата данни.



Фиг.53. Създаване на манипулация в базата данни

id	name	note	price	vat
5	Test	test	0	20

(Screenshot от phpMyAdmin)

Забележка: id-то на манипулацията е 5, защото преди това са избрити някои манипулации от базата данни. Същото важи за пациентите и записаните часове.

Сега създаваме пациента (попълваме само задължителните полета и поставяме състояние „Пломба“ на единия му зъб):

Списък с пациенти

Данни за пациента

Име *:
Test Subject

* Задължително поле.

Телефон:

ЕГН *:
9509101444

* Номерът трябва да е уникален.
Задължително поле.

Пол:
Мъж Жена

Адрес:

Кръвна група:

Резус фактор:

Алергии:

Зъб 18

Текущо състояние:
Пломба

Извършвани манипулации:

Затвори

Забележка:

Запази Нов Изтриване

Фиг.54. Създаване на пациент в базата данни

=> Пациентът е успешно добавен в базата данни.

Списък с пациенти

Test Subject - 9509101444

Showing rows 0 - 0 (1 total, Query took 0.0008 seconds.)

SELECT * FROM 'patient'

id	address	allergies	blood_type	diabetes_status	egn	full_name	gender	heart_fa
4					9509101444	Test Subject	male	

Showing rows 0 - 24 (32 total, Query took 0.0005 seconds.)

SELECT * FROM 'tooth'

id	status	patient_id
97	Пломба	4
98	Здрав	4
99	Здрав	4
100	Здрав	4
101	Здрав	4
102	Здрав	4

Сега следва да създадем график и да поставим манипулацията върху един от зъбите (ще използваме този, на който има пломба):

График подреден по дата и час

Извършени услуги и цени:

☐ Платено (Цена: 0.00 лв.)

Информация за графика

Дата *:

01.01.2018

* Задължително поле.

Избран час *:

00:00

* Комбинацията дата/час трябва да е уникална. Задължително поле.

Пациент *:

Test Subject (9509101444)

* Задължително

Статус на зъб

18 17 16 27 28

48 47 46 37 38

Зъб 18

Текущо състояние:

Пломба

Извършвани манипулации:

Текуща манипулация:

Test

Затвори

Забележка:

Запис Нов Изтриване

=> Графикът (записаният час) е успешно добавен в базата данни.



График подреден по дата и час

(2018-01-01) @ 00:00 - Test Subject

Извършени услуги и цени:

0.00 лв. - Test



Showing rows 0 - 0 (1 total, Query took 0.0006 seconds.)

SELECT * FROM `schedule`

Show all | Number of rows: 25 | Filter rows: Search this table

Options

	id	schedule_date	note	paid_status	price_to_pay	schedule_time	patient_id
	4	2018-01-01		NULL	0	00:00	4



Showing rows 0 - 0 (1 total, Query took 0.0008 seconds.)

SELECT * FROM `tooth_applied_manipulations`

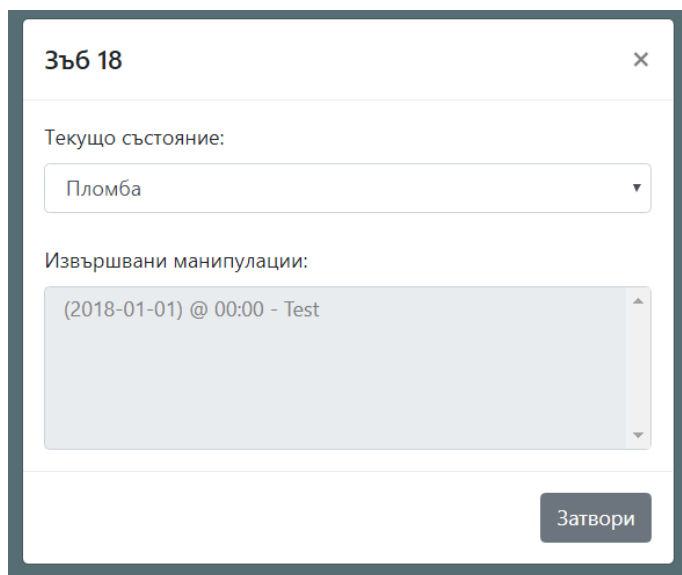
Show all | Number of rows: 25 | Filter rows: Search this table

Options

	manipulated_teeth_id	applied_manipulations_id
	97	5

Фиг.55. Създаване на график в базата данни

Сега виждаме и крайния резултат, когато отворим модалния прозорец за зъб 18 на Test Subject.



Фиг.56. Краен резултат на use case-a

6. Заключение (основни резултати и изводи)

Зъболекар 2.0 предоставя едно просто, но все пак ефективно уеб приложение, което подпомага управлението на една стоматология. Макар на пазара да съществуват множество други решения с по-голям набор от функции, Зъболекар 2.0 залага на простия си и интуитивен дизайн и въпреки всичко върши работа.

Най-големият плюс на Зъболекар 2.0 е, че предоставя ефективен начин за управление на пациентите, статуса на зъбите им (заедно с прилаганите манипулации върху тях) и записаните им часове, а това улеснява работата на стоматолога.

Някои подобрения, които могат да бъдат направени по приложението:

- работа с повече от един стоматолог (въвеждане на акаунти в системата)
- по-добра валидация на данните (показване на съобщение за грешка още при самото въвеждане)
- оптимизация на кода, за да се минимизират нужните ресурси от устройството на потребителя
- възможност за експортиране на информацията от базата данни в други формати (CSV, PDF)

- възможност за импортиране на информация в базата данни от други формати (CSV) и възможност да се импортира БЕЗ да се изтрива предишната информация.

7. Използвана литература

- [1] *Софтуер за "Лекари по дентална медицина" Dentist*. Извлечено 08.2018 от EMG Soft: https://emgsoft.net/scripts/main.php?action=soft_dental
- [2] *Стоматологичен софтуер "Dentistry" - софтуерен продукт за общопрактикуващи стоматолози и специалисти*. Извлечено 08.2018 от Лабиринт 05: <http://www.labyrinth05.com/index.php/software1/s1>
- [3] *PracticeDent - удобна и богата на функционалности система за управление на денталната практика*. Извлечено 08.2018 от Practicedent.com: <https://www.practicedent.com/bg/index.html#features>
- [4] *Безплатен стоматологичен софтуер Dental.bg*. Извлечено 08.2018 от Dental.bg <https://dental.bg/clinicpages/72/bezplaten-stomatologichen-softuer>
- [5] *Spring Framework*. Извлечено 08.2018 от Wikipedia: https://bg.wikipedia.org/wiki/Spring_Framework
- [6] Karanam, Ranga Rao (1 февруари, 2017). *Spring Boot vs Spring MVC vs Spring - How do they compare?* Извлечено 08.2018 от SpringBootTutorial.com: <http://www.springboottutorial.com/spring-boot-vs-spring-mvc-vs-spring>
- [7] *mysql-backup4j - a library for programmatically exporting mysql databases*. Извлечено 08.2018 от GitHub: <https://github.com/SeunMatt/mysql-backup4j>
- [8] *jQuery*. Извлечено 08.2018 от Wikipedia: <https://bg.wikipedia.org/wiki/JQuery>
- [9] *pdfmake - Client/server side PDF printing in pure JavaScript*. Извлечено 08.2018 от GitHub: <https://github.com/bpampuch/pdfmake>
- [10] *Why does Hibernate require no argument constructor?* Извлечено 08.2018 от StackOverflow: <https://stackoverflow.com/questions/2935826/why-does-hibernate-require-no-argument-constructor>
- [11] *GenerationType.AUTO vs GenerationType.IDENTITY in Hibernate*. Извлечено 08.2018 от StackOverflow: <https://stackoverflow.com/questions/33096466/generationtype-auto-vs-generationtype-identity-in-hibernate>

- [12] *What's the difference between @JoinColumn and mappedBy when using a JPA @OneToMany association.* Извлечено 08.2018 от StackOverflow:
<https://stackoverflow.com/questions/11938253/whats-the-difference-between-joincolumn-and-mappedby-when-using-a-jpa-onetoma>
- [13] *Interface CrudRepository<T,ID> documentation.* Извлечено 08.2018 от Spring.io:
<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>
- [14] *Query Creation.* Извлечено 08.2018 от Spring.io:
<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.query-creation>
- [15] *CSS - Global CSS settings, fundamental HTML elements styled and enhanced with extensible classes, and an advanced grid system.* Извлечено 08.2018 от GetBootstrap.com:
<https://getbootstrap.com/docs/3.3/css/>
- [16] *Cards.* Извлечено 08.2018 от GetBootstrap.com:
<https://getbootstrap.com/docs/4.0/components/card/>
- [17] *Navbar - Documentation and examples for Bootstrap's powerful, responsive navigation header, the navbar.* Извлечено 08.2018 от GetBootstrap.com:
<https://getbootstrap.com/docs/4.0/components/navbar/>
- [18] *Form groups.* Извлечено 08.2018 от GetBootstrap.com:
<https://v4-alpha.getbootstrap.com/components/forms/#form-groups>
- [19] *Collapse.* Извлечено 08.2018 от GetBootstrap.com:
<https://getbootstrap.com/docs/4.1/components/collapse/>
- [20] *Modal.* Извлечено 08.2018 от GetBootstrap.com:
<https://getbootstrap.com/docs/4.0/components/modal/>
- [21] *Script inlining (JavaScript and Dart).* Извлечено 08.2018 от Thymeleaf.org:
<https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html#script-inlining-javascript-and-dart>
- [22] *pdfmake – Getting Started.* Извлечено 08.2018 от Pdfmake.org:
<http://pdfmake.org/#/gettingstarted>
- [23] *Navs – JavaScript behavior.* Извлечено 08.2018 от GetBootstrap.com:
<https://getbootstrap.com/docs/4.0/components/navs/#javascript-behavior>
- [24] *Change default text in input type="file"?* Извлечено 08.2018 от StackOverflow:
<https://stackoverflow.com/questions/5138719/change-default-text-in-input-type-file>