

## **ДИПЛОМНА РАБОТА**

ОБРАЗОВАТЕЛНА СТЕПЕН БАКАЛАВЪР ПО КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ

**Тема:** Разработване на автоматизирана система за контрол на качеството на печатни платки при производството на електронни изделия.

**Изготвил:**

**Петко Станимиров Арбов**

**Фак. № 121209014**

**Ръководител:**

**инж. Симеон Цветанов**

**София**

**2013**



## Съдържание:

1. Увод .....	5
2. Постановка на дипломната работа, цели и задачи .....	6
2.1 Постановка на проблема .....	6
2.2 Цел на дипломната работа .....	6
2.3 Задачи .....	6
3. Описание на наличната инфраструктура .....	8
4. Проектиране на системата .....	10
4.1 База данни .....	10
4.1.1 Структура на базата данни .....	11
4.1.2 Таблици .....	11
4.1.3 Връзки .....	13
4.1.4 Индекси .....	13
4.2 Библиотека за достъп до базата данни .....	13
4.3 Приложение за попълване на базата данни .....	13
4.4 Сървърно приложение .....	14
4.4.1 Обработка на заявките (requests) .....	14
4.4.2 Работа с потребители .....	14
4.4.3 Заявки за информация за печатни платки .....	14
4.4.4 Модели за сериализация на данните .....	15
4.5 Клиентско приложение .....	15
4.5.1 Създаване и обработка на заявки към сървъра .....	15
4.5.2 Създаване на доклади .....	16
4.5.3 Потребителски интерфейс .....	16
5. Програмна реализация .....	17
5.1 Използвани софтуерни технологии .....	17
5.1.1 Операционна система Windows .....	17
5.1.2 Език за програмиране C# .....	17
5.1.3 Среда за разработка Visual Studio .....	17
5.1.4 Релационна база данни MS SQL Server Express .....	17
5.1.5 Web API .....	17
5.1.6 .Net Framework 4.5 .....	17
5.1.7 Entity Framework .....	18
5.2 База Данни .....	18
5.3 Библиотека за достъп до базата данни .....	18
5.3.1 Entity Framework Database model .....	18
5.3.2 LINQ to SQL .....	19
5.3.3 Класът PcbDAL .....	20
5.4 Приложение за попълване на данни в базата .....	20
5.5 Сървърно приложение .....	21
5.5.1 Структура на приложението .....	21
5.5.2 Контролер BaseApiController .....	22
5.5.3 Контролер UsersController .....	23
5.5.4 Контролер PcbcsController .....	24
5.5.5 Модели .....	37
5.5.6 HTTP routes .....	38
5.6 Клиентско приложение .....	39
5.6.1 Организация на приложението .....	39
5.6.2 Хеширане на паролата .....	40

5.6.3 Комуникация със сървъра .....	40
5.6.4 Генериране на доклади.....	41
5.6.5 Прозорци на приложението.....	41
5.6.5.1 Вход в системата.....	41
5.6.5.2 Основен прозорец .....	42
5.6.5.3 Прозорец за визуализиране на информацията за платка.....	42
5.6.5.4 Прозорец за генериране на доклад по дата .....	44
5.6.5.5 Прозорец за генериране на доклад по дата и оператор .....	46
5.6.5.6 Прозорец за генериране на доклад по поръчка.....	47
6. Ръководство за работа с приложението .....	48
6.1 Внедряване на системата .....	48
6.2 Работа с клиентското приложение.....	48
7. Заключение.....	50
8. Литература.....	51

## **1. Увод**

С нарастването на производствените мощности, увеличаването на работния персонал, увеличаване на гамата продукти във всяко производство възниква нужда от начин за съхраняване и извличане на информация и статистики, генерирани по време на производствения процес. Появява се необходимост от контрол върху заетостта на персонала, производствените мощности, с които предприятието работи и др.

Така се появява нуждата от системи, които подпомагат управлението на бизнеса. Те могат да спомогнат за стабилизация на бизнеса чрез осигуряване на точна и подредена информация и предоставяне на достъп до нея по всяко време и от всяко място

В настоящата дипломна работа е описано създаването на интегрирана система за събиране на данни при производството на печатни платки. Целта на системата е да систематизира данните, получени по време на процеса на произвеждане на печатната платка, като предоставя възможност за контрол чрез система за репорти.

## **2. Постановка на дипломната работа, цели и задачи**

### **2.1 Постановка на проблема**

Основен елемент на серийно произвеждано електронно изделие е печатна платка, управлявана от чип - компютър. След насищането на елементите по платката се преминава към запис на програмата в чип компютъра. Този процес се извършва с помощта на програма за запис, която се изпълнява на специален стенд. В зависимост от заявката за производство на платките, преди запис на програмата в платката се въвеждат серия параметри, които имат стойност при понататъшна употреба на платката. Информацията за записаните в платката програма и параметри се запазва в текстови файл. След успешен запис на програмата в платката се преминава към тестове. Стойностите на параметрите, които се проверяват по време на тестването се запазват в текстови файл. След успешен запис и тест, платката се предава за асемблиране в устройство и се преминава към калибровка и крайни изпитания.

Данните, записани в тези текстови файлове се използват в процеса на поддръжка на електронните изделия. Това създава необходимост от централизирана система, която да съхранява тези данни и при нужда да извлича необходимата информация.

### **2.2 Цел на дипломната работа**

С използване на така натрупаните резултати от процеса, описан в точка 2.1, се разработва система, чиято цел е да се осигури цялостно информационно обслужване на процеса на текущото изпълнение на всяка заявка за производство, натовареност на персонала, предоставяне на статистическа информация за качеството на отделни възли (предимно сензори, които са собствено производство), предоставяне на цялата налична информация за печатните платки, техните характеристики и създаване на система от доклади, които следят изпълнението на поръчките и график на производството.

### **2.3 Задачи**

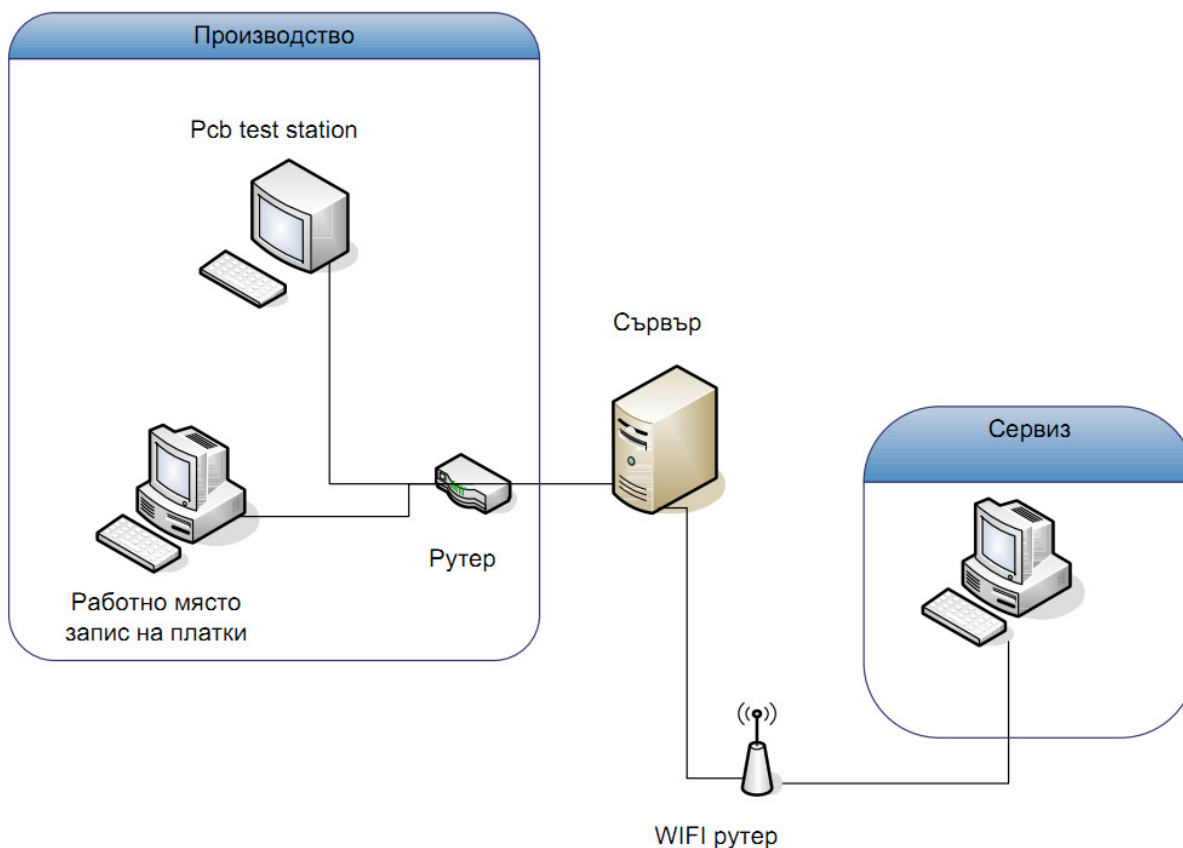
Настоящата дипломна работа реализира следните задачи:

- Разработване на база данни за съхранение на информацията за платка, получена по време на производствения процес;
- Възможност за обработка на архивните файловете от работните станции и добавяне на информацията от тях в базата данни;
- Разработване на сървърно приложение, предоставящо информация от базата данни;
- Разработване на Desktop клиентско приложение за обработка на предоставяната от сървъра информация;
- Вход в системата чрез потребителско име и парола;
- Възможност за извличане на цялата налична информация за конкретна печатна платка;

- Възможност за извличане на специфична информация за всяка печатна платка, генерирана на работно място;
- Възможност за създаване на справки за произведените печатни платки по критерии: дата на производство, номер на поръчка;
- Възможност за генериране на репорти, с цел следене производствените мощности, заетостта на персонала;
- Уведомяване при изход от системата.

### 3. Описание на наличната инфраструктура

На фигура 3.1 е показана схема на наличната информационна инфраструктура.



фигура 1.1 Схема на налична информационна структура

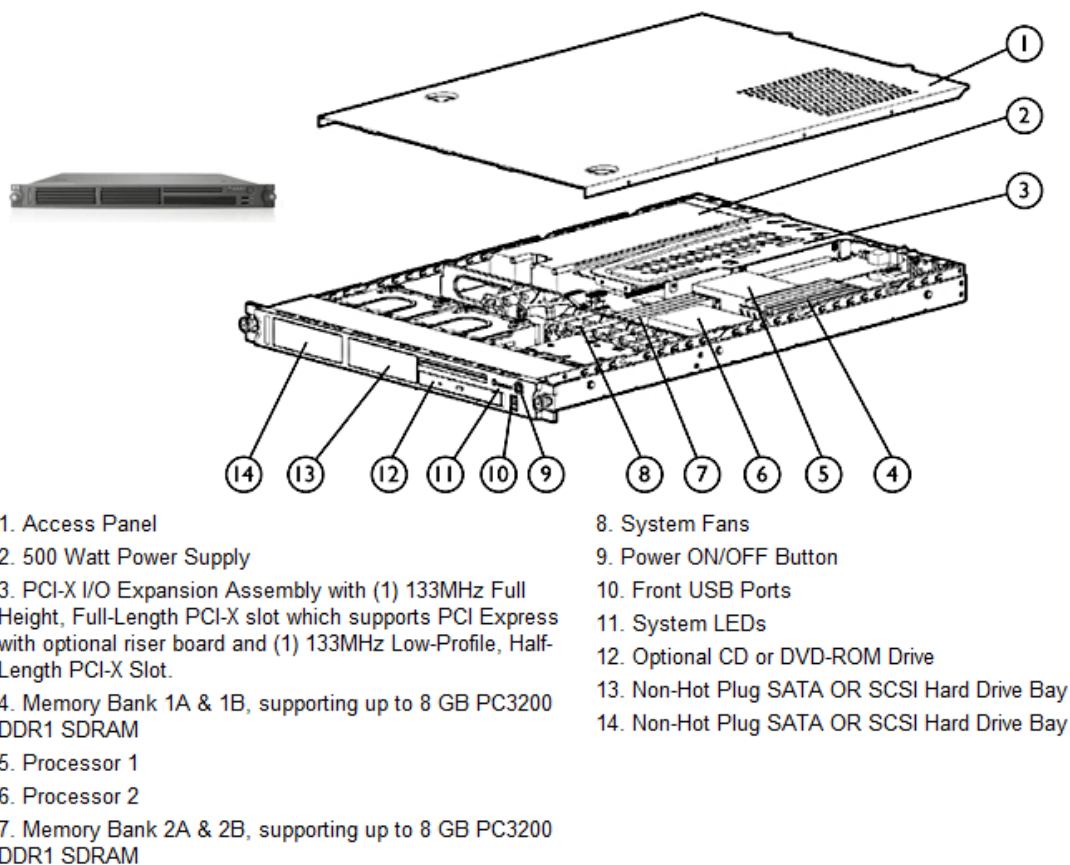
Показаната на фигура 3.1 информационна инфраструктура не е пълно описание на наличната, а показва елементите, които засягат или биха засегнали системата.

Фирмата има изградена единна вътрешна компютърна мрежа. Всички офиси имат свързаност към тази вътрешна мрежа. В центъра на компютърната мрежа стои сървър, на който се съхраняват използваните програми по работните места и на който се пазят файловете с резултатите от извършените дейности. Сървърът работи под операционна система Microsoft Windows Server 2008 (включваща .NET Runtime) и има инсталиран на него Microsoft SQL Server 2012 Express.

Сървърът е марка HP, модел ProLiant DL145. Техническите характеристики са дадени на фигура 3.2 и таблица 1.



## HP ProLiant DL145



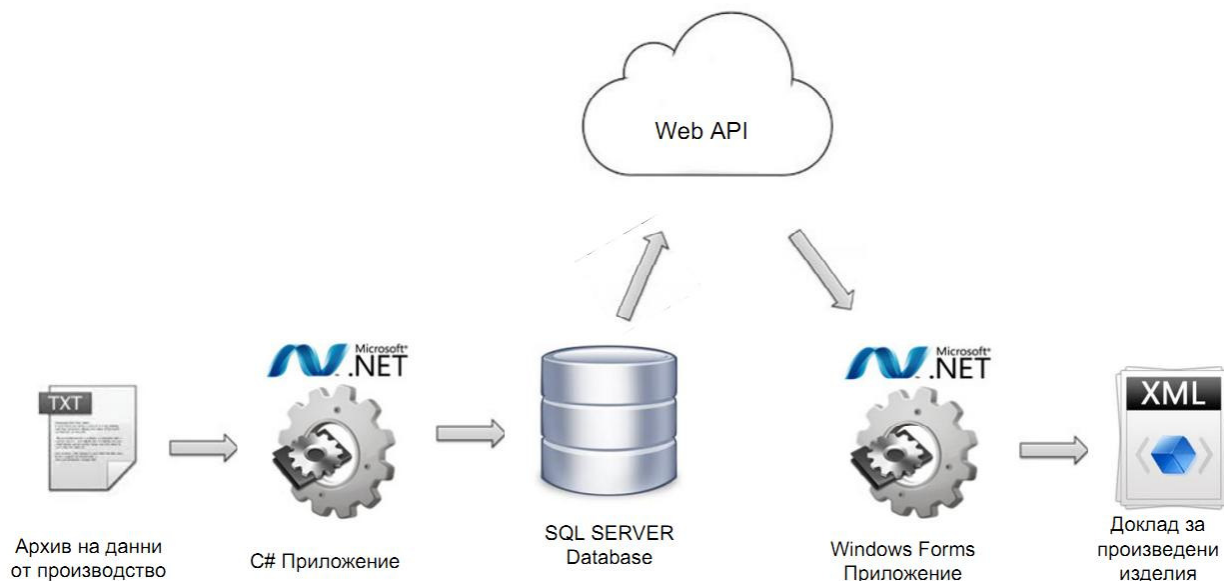
фигура 2.2 Технически характеристики на сървъра

	» <a href="#">DL145</a>
<b>Processor</b>	AMD Opteron™ 2GHz 1MB L2 cache 800 MHz HyperTransport
<b>Chipset</b>	AMD 8111 and AMD 8131
<b>RAM std/max</b>	4GB
<b>Memory technology</b>	PC2700 ECC DDR SDRAM @ 333MHz
<b>Drive controller</b>	Integrated dual-channel ATA
<b>RAID controller</b>	Optional PCI card
<b>NIC</b>	Integrated dual Broadcom 5704 10/100/1000
<b>Hard drive bays</b>	2 non-hot plug ATA or SCSI
<b>Slots</b>	1 × 133MHz PCI-X
<b>Management</b>	IPMI 1.5
<b>Video</b>	Integrated ATI Rage XL @ 8MB
<b>Removable media</b>	2 USB ports (1 front & 1 rear)
<b>Warranty</b>	1-0-0

Таблица 1 Технически характеристики на сървъра

## 4. Проектиране на системата

За постигане на целите на дипломната работа са необходими релационна база данни, клас библиотека, която осигурява достъпа до данните в базата данни, приложение, което обработва производствените лог файлове и попълва информацията в базата данни, сървърно и клиентско приложение. За връзка ще се използва световната мрежа Internet и протоколът HTTP. Обобщена схема на системата е показана на фигура 4.1

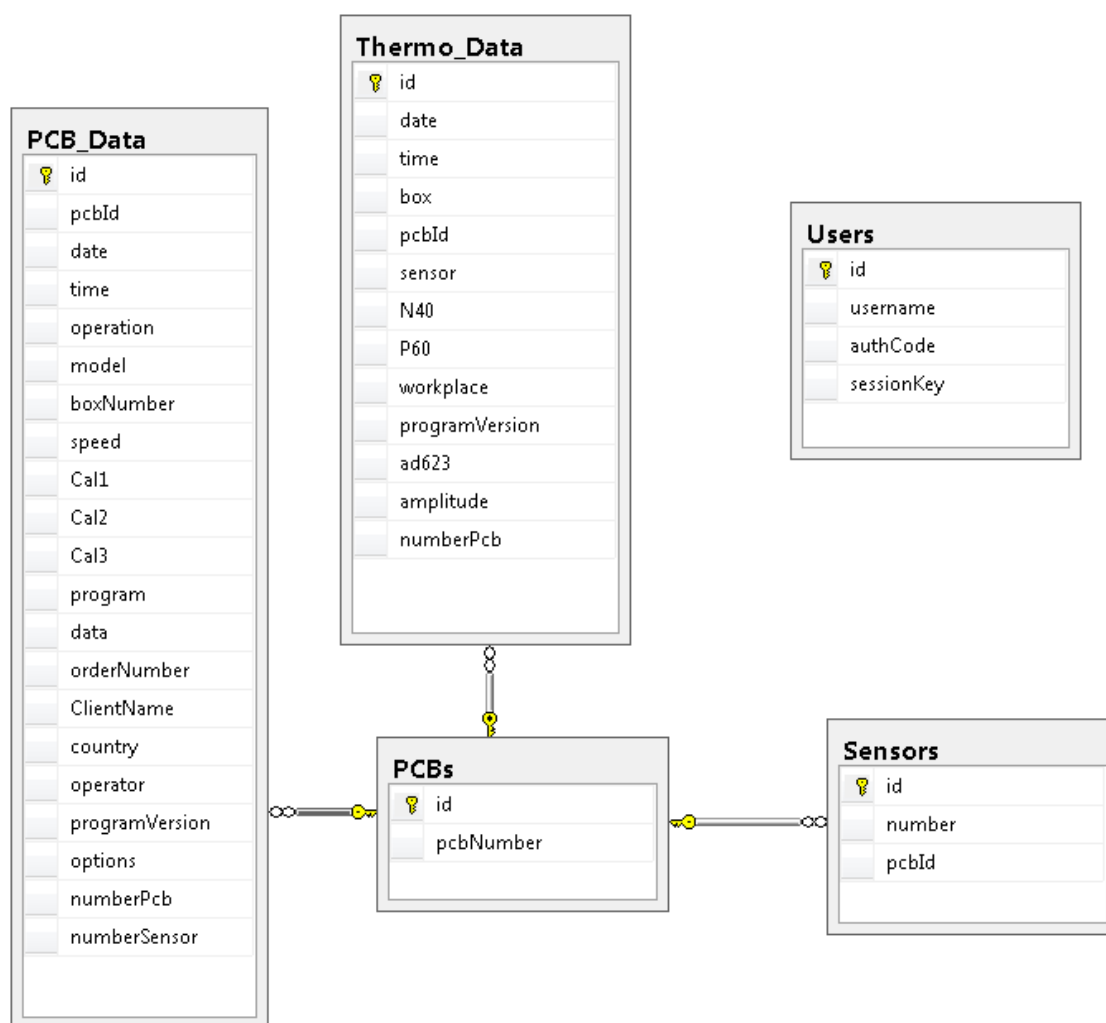


фигура 4 Обобщена схема на системата

### 4.1 База данни

Базата данни има за цел да организира и съхранява генерираната информация в процеса на запис, настройка и тестване на печатните платки.

### 4.1.1 Структура на базата данни



фигура 4.1 Структура на базата данни

Структурата на базата данни се състои от 4 таблици, в които се запазва информацията за печатните платки и една таблица, в която се запазват данните за потребителите, които могат да имат достъп до системата

### 4.1.2 Таблици

Таблицата **PCBs** стои в центъра на базата данни. Тя е основен елемент на така създаваната система. Цялата събрана информация в процеса на запис, настройка и тест е обвързана с печатната платка. Таблицата се състои от следните полета:

- Id – тип integer, auto-increment, служи за първичен ключ в таблицата
- pcbNumber – тип integer, запазва номера на платката

Таблицата **PCB\_Data** съхранява информацията от лог файла при първоначален запис и настройка на платката. Таблицата се състои от следните полета:

- Id – тип integer, auto-increment, служи за първичен ключ в таблицата

- pcbId – тип integer, служи за външен ключ в таблицата за връзка с таблица PCBs
- date – тип nchar(10), запазва се датата на запис на платката
- time – тип nchar(10), запазва се часът на запис на платката
- operation – тип nvarchar(50), запазва операциите за запис
- model – тип nchar(10), запазва модела на платката
- boxNumber -тип int, запазва номера на кутията, в която ще бъде поставена платката
- speed –тип nvarchar(50), запазва скоростта на сензора за едно измерване
- Cal1 – тип nvarchar(15), запазва първата калибровка на сензора
- Cal2 - тип nvarchar(15), запазва втората калибровка на сензора
- Cal3 - тип nvarchar(15), запазва третата калибровка на сензора
- Program – тип varchar(max), запазва името на програмата за запис
- Data – тип varchar(max), запазва данните за запис в платката
- orderNumber – тип nvarchar(10), запазва номера на поръчката, от която е част платката
- ClientName – тип nvarchar(50), запазва името на клиента
- Country - тип nvarchar(50), запазва името на държавата, за която се произвежда платката
- Operator – тип nchar(10), запазва инициалите на оператора, извършил записа на платката
- programVersion – тип float, запазва версията на програмата за запис
- options – тип nvarchar(50), запазва списък на допълнителните опции на платката
- numberPcb – тип int, запазва номера на платката
- numberSensor – тип int, запазва номера на сензора, вграден в платката

Таблицата **Thermo\_Data** съхранява информацията от лог файловете при тестване на вече записаната платка. Таблицата се състои от следните полета:

- Id – тип int, auto-increment, служи за първичен ключ в таблицата
- Date – тип nvarchar(50), запазва датата на тест на платката
- Time – тип nchar(50), запазва часът на тестване на платката
- Box – тип int, запазва номера на кутията, в която ще бъде монтирана платката. Възможна е разлика със запазения номер на кутия в предходната таблица
- pcbId – тип int, външен ключ към таблица PCBs
- sensor – тип int, запазва номера на сензора
- N40 – тип int, запазва стойността на параметър N40 на сензора
- P60 – тип int, запазва стойността на параметър P60 на сензора
- Workplace – тип nvarchar(50), запазва номера на работното място на което е извършен теста
- programVersion – тип float, запазва версията на програмата с която е извършен теста
- ad623 – тип int, запазва стойността на параметъра ad623 на сензора
- amplitude – тип int, запазва стойността на амплитудата на сигнала на сензора
- numberPcb – тип int, запазва номера на платката. Може да има промяна от стойността, запазена в предната таблица

Таблицата **Sensors** съхранява информацията за сензора, който е инсталиран в конкретната платка. Таблицата се състои от следните полета:

- Id – тип int, първичен ключ на таблицата
- Number – тип int, запазва номера на сензора
- pcbId – тип int, външен ключ, сочи към таблица PCBs

Таблицата **Users** съхранява данните за хората, които ще имат достъп до системата. Таблицата се състои от следните полета:

- Id – тип int, първичен ключ на таблицата

- Username – тип nvarchar(50), запазва потребителското име, с което се авторизира потребителя
- authCode – тип nvarchar(40), запазва хешираната парола на потребителя
- sessionKey – тип nvarchar(50), запазва генерирания след вход в системата сесиен ключ

#### 4.1.3 Връзки

- **PCBs-Sensors** – Връзката е „едно към много”. Принципно в една платка има само един сензор, но в процеса на тестване може да се окаже че сензора е повреден и да бъде сменен. Затова е възможно повече от един сензор да сочи към конкретна платка
- **PCBs-PCB\_Data** - Връзката е „едно към много”. Поради възможността за смяна на сензора и презаписване на информацията в платката е възможно повече от един запис в таблица PCB\_Data да сочи към конкретна платка
- **PCBs-Thermo\_Data** - Връзката е „едно към много”. Поради възможността за смяна на сензора и преминаване отново през тестовите на платката е възможно повече от един запис в таблица Thermo\_Data да сочи към конкретна платка

#### 4.1.4 Индекси

- **IX\_FK\_PCB\_Data\_PCBs** (Non-unique, Non-Clustered) – външен ключ от таблица PCB\_Data, сочещ към главния ключ на таблица PCBs
- **PK\_PCB\_Data**(Clustered) – главен ключ на таблица PCB\_Data
- **UK\_PCBs\_PcbNumber** (Unique, Non-Clustered) – индекс, гарантиращ уникалност на всяка стойност в колоната pcbNumber в таблица PCBs
- **PK\_PCBs** (Clustered) – главен ключ на таблица PCBs
- **UK\_Sensors\_Number** (Unique, Non-Clustered) – индекс, гарантиращ уникалност на всяка стойност в колоната number в таблица Sensors
- **IX\_FK\_Sensors\_PCBs** (Non-Unique, Non-Clustered) – външен ключ на таблица Sensors, сочещ към главния ключ на таблица PCBs
- **PK\_Sensors** (Clustered) – главен ключ на таблица Sensors
- **PK\_Thermo\_Data** (Clustered) – главен ключ на таблица Thermo\_Data
- **PK\_Users** (Clustered) – главен ключ на таблица Users
- **UK\_Users\_username** (Unique, Non-Clustered) – индекс, гарантиращ уникалност на записите в колоната username на таблица Users

## 4.2 Библиотека за достъп до базата данни

Системата се състои от три приложения, които обработват данни и четат или пишат в базата данни. За улеснение се създава слой за достъп до базата данни (Data Access Layer). По този начин, когато на приложение от системата му трябва връзка до базата данни, то се „обръща” към този слой. Слойът представлява библиотека класове (class library), в която чрез Entity Framework релационната база данни се създава модел от обекти(класове и асоциации).

Създава се Entity Framework model на релационната база данни, контекст за достъп до данните чрез LINQ to SQL и се създава клас, който се състои от функции за попълване на данни в базата.

## 4.3 Приложение за попълване на базата данни

Всяко работно място по пътя на платката в системата за запис, настройка и тестване създава архивен файл на резултатите от своята дейност. Архивните файлове (.log) са с ясна структура и се знае коя информация в тях може да бъде пропусната в процеса на създаване на файла. Създават се функции за изчитане на архивните файлове (парсване) и предаването на данните от тях към слоя за достъп до данните, описан в точка 4.2, който ги записва в базата данни.

## 4.4 Сървърно приложение

За тип на сървърното приложение е избран ASP .NET Web API. Това е платформа, позволяваща лесно изграждане на HTTP услуги за браузъри, мобилни и десктоп клиенти. Използва се за създаване на RESTful приложения в платформата .Net. Приложението работи на принципа на контролери за данни.

### 4.4.1 Обработка на заявките (requests)

Сървърът трябва да може да обработва заявките, които пристигат от клиентите. Създава се специален клас (контролер), който в зависимост от постъпилата заявка за действие, връща специфична информация. Създава се по една функция за всяко действие (action).

### 4.4.2 Работа с потребители

Сървърното приложение трябва да пази потребителите в системата, които са регистрирани, и да позволява вход в системата чрез име и парола. Данните за потребителите се пазят в релационната база данни, описана в точка 4.1. За всеки потребител се пазят:

- ID – число, идентификационен номер, първичен ключ
- Потребителско име – символен низ, с който се идентифицират потребителите
- Парола – символен низ, секретна дума, която да удостоверява автентичността на потребител
- Сесиен ключ – символен низ, състоящ се от букви и цифри, който се генерира при влизане на потребител в системата и се подава при всяка заявка към сървърното приложение за удостоверяване на потребителя, извършващ заявката. След изход от системата сесийния ключ се нулира

Потребители се създават само от администратора на базата данни. Няма публична регистрация на потребители.

За по-лесна работа с данните на потребителите се създава отделен контролер, който да обработва тези данни. Така ако се наложи да се променя нещо, всички функции за работа с потребители ще бъдат в този контролер. Ще се реализират следните функции

Функция	Действие (action)	Метод	Описание
Login	“login”	HTTP POST	Вход в системата, генериране на сесиен ключ
Logout	“logout”	HTTP PUT	Изход от системата

Функциите в този контролер изпълняват т.н. бизнес логика. Благодарение на отделянето им, лесно може да се променя част от логиката, източника на базата данни и др.

### 4.2.3 Заявки за информация за печатни платки

Сървърното приложение трябва да извлича информация за печатните платки, които се пазят в релационната база данни, описана в точка 4.1

За по-лесна работа с извлечените данни се създава отделен контролер. Ще се реализират следните функции:

Функция	Действие (action)	Метод	Описание
GetAll	"get-all"	HTTP GET	Връща цялата налична информация за всички платки
GetByPcbNumber	"get-by-pcb"	HTTP GET	Връща наличната информация за конкретна платка
GetByDate	"get-by-date"	HTTP GET	Връща списък от платки, записани на конкретна дата

GetByDateAndOperator	"get-by-date-operator"	HTTP GET	Връща списък от платки, записани на конкретна дата и от конкретен оператор
GetByOrderNumber	"get-by-order"	HTTP GET	Връща списък от платки към дадена поръчка
GetAllPcbData	"get-pcbdata"	HTTP GET	Връща цялата налична информация от единия лог файл
GetPcbDataByNumber	"get-pcbdata-bynumber"	HTTP GET	Връща наличната информация за конкретна платка от работно място pcb tester
GetPcbDataByDate	„get-pcbdata-bydate"	HTTP GET	Връща списък от платки, записани на конкретна дата от работно място pcb tester
GetPcbDataByOrder	"get-pcbdata-byorder"	HTTP GET	Връща списък от платки към дадена поръчка от работно място pcb tester
GetAllThermoData	"get-thermodata"	HTTP GET	Връща списък от платки, минали тестовете
GetThermoDataByNumber	"get-thermodata-bynumber"	HTTP GET	Филтрира списъка от горната заявка по номер на платка
GetThermoDataByDate	"get-thermodata-bydate"	HTTP GET	Филтрира списъка от GetAllThermoData по дата

Отделянето на функциите в контролер позволява лесна промяна във функционалността, източника на базата данни и др.

#### 4.2.4 Модели за сериализация на данните

В ASP .Net Web API моделите се използват за сериализация и десериализация на данните. Сървърното приложение връща данни и получава данни в JSON (JavaScript object notation) формат. Създават се модели, по които се сериализира информацията, която приложението връща в отговор на заявката и модели за десериализиране на информацията, подадена при заявката към сървърното приложение.

### 4.5 Клиентско приложение

Клиентското приложение трябва да е Windows програма с графичен потребителски интерфейс и приятна, ненаатрапчива външност.

#### 4.5.1 Създаване и обработка на заявки към сървъра

Клиентското приложение трябва да използва сървисите, предоставени от сървърното приложение и по адекватен начин за обработка получените резултати и да визуализира данните.

Създават се модели за десериализация на данните, върнати като отговор на заявката от сървъра в JSON формат.

Създават се модели за сериализация на данните, предавани като параметри в заявките към сървъра.

#### **4.5.2 Създаване на доклади**

Реализира се функционалност за архивиране на данните, получени в отговор на заявките към сървъра в XML формат.

Създават се функции за генериране на репорти по различни критерии като:

- Дата на производство
- Дата на производство и оператор, извършил записа или тестването на платката
- Номер на поръчка

#### **4.5.3 Потребителски интерфейс**

Графичният потребителски интерфейс ще представлява комбинация от прозорци, бутони, текстови полета и други. Важно за такива приложения е те да имат привлекателен и интуитивен характер.

Приложението трябва да има следните прозорци:

- Вход в системата
- Основен прозорец на приложението
- Визуализация на цялата налична информация за конкретна платка
- Задаване на критерии за генериране на репорт

При писане на текст в полетата трябва да се реализира адекватна валидация на данните и да не се позволяват злоупотреби. Визуализация на съобщения за грешка в процеса на комуникация със сървъра. Уведомление при вход и изход в системата.



## **5. Програмна реализация**

### **5.1 Използвани софтуерни технологии**

#### **5.1.1 Операционна система Windows**

Windows е фамилия операционни системи, произвеждани от компанията Microsoft. Те са доминиращи на пазара на компютри за крайни потребители и печелят позиции на пазара на сървърни операционни системи. В настоящата дипломна работа ще се използва Windows, както за клиентското приложение, така и за сървърното. Поддържа се версия Windows XP или по-нова. Конкретното приложение е разработвано на Windows 7 Professional Edition

#### **5.1.2 Език за програмиране C#**

C# е съвременен, обектно-ориентиран и типово обезопасен език за програмиране, който е наследник на C и C++. Той комбинира леснотата на използване на Java с мощността на C++. Създаден от екипа на Андере Хейлсбърг, архитектът на Delphi, C# заимства много от силните страни на Delphi- свойства, индексатори, компонентна ориентираност. C# въвежда и нови концепции – разделяне на типовете на два вида – стойностни (value types) и референтни (reference types), автоматично управление на паметта, делегати и събития, атрибути, XML документация и други. Той е стандартизиран от ECMA и ISO.

C# е специално проектиран за .Net Framework и е съобразен с неговите особености. Той е сравнително нов, съвременен език, който е заимствал силните страни на масово използваните езици за програмиране от високо ниво, като C, C++, Java, Delphi, PHP и др. [2]

#### **5.1.3 Среда за разработка Visual Studio**

Visual studio е интегрирана среда за разработка на софтуер, предлагана от компанията Microsoft. Налични са всички необходими инструменти за разработка на приложения за Windows операционна система. Ще се използва Visual Studio 2012 Professional

#### **5.1.4 Релационна база данни MS SQL Server Express**

Microsoft SQL Server Express, е безплатна версия на релационната база данни Microsoft's SQL Server, която представлява база данни, насочена към приложения за дребния бизнес.

#### **5.1.5 Web API**

Сървърната страна на web API е програмен интерфейс към дефинирана система, базирана на заявка-отговор, обикновено представена в JSON или XML формат, който е достъпен през мрежата, най-често чрез уеб базиран HTTP сървър..<sup>[6]</sup>

"Web API" в този контекст понякога е смятан за синоним за уеб услуга, Web 2.0 web applications се преместиха от service-oriented architecture (SOA) със SOAP-базирани уеб услуги към колекции от RESTful web resources. Тези RESTful web APIs са достъпни през стандартни HTTP методи и от голяма гама HTTP клиенти, включително браузъри и мобилни устройства.

#### **5.1.6 .Net Framework 4.5**

.Net Framework е библиотека от класове, която може да се инсталира на компютри с операционна система Windows. Съдържа се виртуална машина, която изпълнява програмния код. .Net

Framework има много положителни качества и се препоръчва от Microsoft за разработка на нови приложения. В настоящата дипломна работа ще се използва версия 4.5, която е последната версия на библиотеката, която е необходима за правилното функциониране на програмите

### **5.1.7 Entity Framework**

Entity Framework е стандартна ORM платформа, част от .Net. Тя осигурява инфраструктура за достъп до данни от база данни като обекти от .Net средата.

Схемата на базата данни се преобразува до модел от обекти. Позволява LINQ заявки към базата данни и има вградени CRUD операции

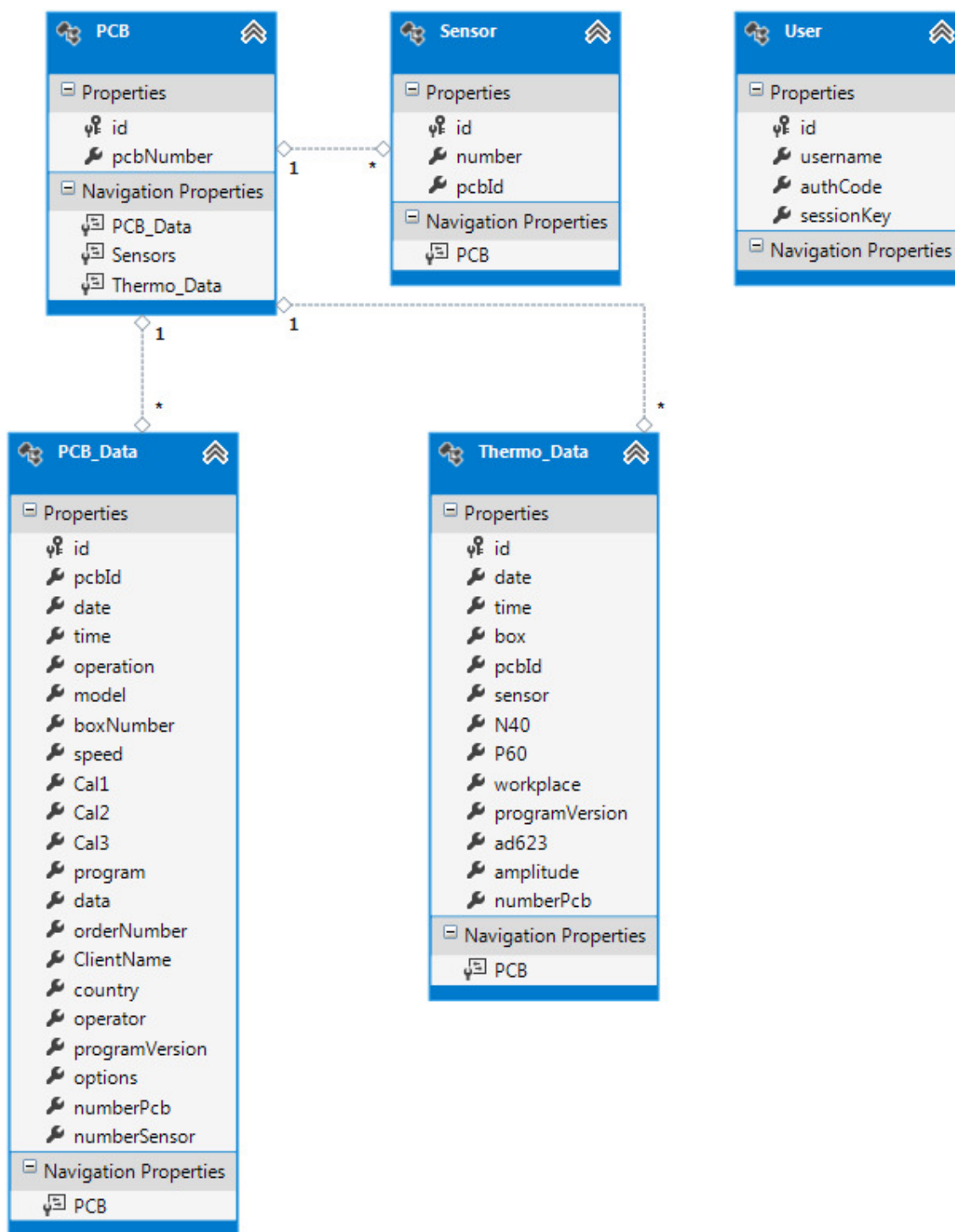
## **5.2 База Данни**

За реализацията на базата данни е използван “database first” подхода. Реализиран е модела от глава 4.1.1 чрез SQL Server Management Studio

## **5.3 Библиотека за достъп до базата данни**

### **5.3.1 Entity Framework Database model**

С помощта на Entity Framework релационната база данни е преобразувана до обекти, които са част от платформата .Net. Моделът на базата данни е показан на фигура 5.1



фигура 5.1 Структура на генерираните обекти и модела на базата данни

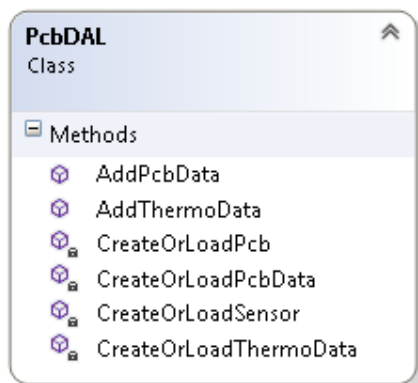
### 5.3.2 LINQ to SQL

За достъп и модификация на базата данни няма да се използват стандартни SQL заявки, а технологията LINQ. Зад абривиатурата LINQ стои Language Integrated Query. Това е проект на Майкрософт за добавяне на синтаксис за заявки, характерен за T-SQL, в езиците от .Net Framework. LINQ дефинира стандартни оператори за заявки, които позволяват на езиците с

поддръжка на LINQ да филтрират, изброяват и създават проекции на няколко типа колекции, като използват единен синтаксис. За създаване на обектите, които ще манипулират базата данни се използва технологията LINQ to SQL, която прави контекст за изпълняване на LINQ заявки. Работата с базата данни е изнесена в класа PcbDAL

### 5.3.3 Класът PcbDAL

Класът PcbDAL (data access layer) се използва за попълване на базата данни. В него се съдържат методи, които получават данни и ги попълват в базата. Методите на класа са показани на фигурата:



фигура 5.2 Диаграма на клас PcbDAL

Методът **AddPcbData** създава контекст на базата данни и приема като параметри стойности, които ще се запишат в таблицата PCB\_Data.

Методът **AddThermoData** създава контекст на базата данни и приема като параметри стойности, които ще се запишат в таблицата Thermo\_Data.

Метод **CreateOrLoadPcb** получава като параметри контекст към базата данни, номер на платката и номер на сензора. Прави се проверка дали съществува запис на платка с такъв номер и ако съществува, се връща този запис. Ако не съществува се създава нов запис. Тази проверка се прави с цел да се гарантира уникалността на номерата на платките, които се пазят в базата. В базата не трябва да се допускат платки с дублирани номера.

Метод **CreateOrLoadSensor** получава като параметри контекст към базата данни, номер на платката и номер на сензора. Прави се проверка дали съществува запис на сензор с такъв номер и ако съществува, се връща този запис. Ако не съществува се създава нов запис. Тази проверка се прави с цел да се гарантира уникалността на номерата на сензорите, които се пазят в базата. В базата не трябва да се допускат сензори с дублирани номера.

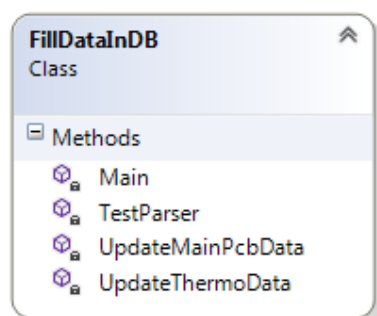
Методът **CreateOrLoadPcbData** приема като параметри контекст на базата данни и стойности за полетата на таблица PCB\_Data от базата данни. Прави се проверка дали съществува запис в таблицата за платка с такъв номер. Ако не съществува се създава нов запис. Проверката се пази с цел недопускане на дублиращи се редове в базата за една и съща платка.

Методът **CreateOrLoadThermoData** приема като параметри контекст на базата данни и стойности за полетата в таблица Thermo\_Data. Прави се проверка дали съществува такъв запис вече и ако не, се създава нов.

## 5.4 Приложение за попълване на данни в базата

Приложението има за цел прочитането на производствените архивни файлове и предаване на прочетените данни към слоя за достъп до базата данни, описан в точка 5.3.

Структурата на приложението е показана на фигурата



фигура 5.3 Диаграма на клас `FillDataInDB`

Файловете са с дефинирана структура, като записите на ред са разделени с табулация. При прочитане на ред от конкретен файл се знае на коя позиция стои информацията за конкретна клетка в таблиците в базата данни. Понеже файловете е възможно да са големи по размер, за прочитането им се използва `StreamReader`.

Методите `UpdateThermoData` и `UpdateMainPcbData` реализират следната функционалност:

- Отварят файла със `StreamReader`
- Четат документите ред по ред
- Всеки прочетен ред се парсва по зададени критерии
- Прочетените данни от всеки ред се предават към функцията за обработка на конкретния тип данни към слоя за достъп до базата данни, описан в точка 5.3

## 5.5 Сървърно приложение

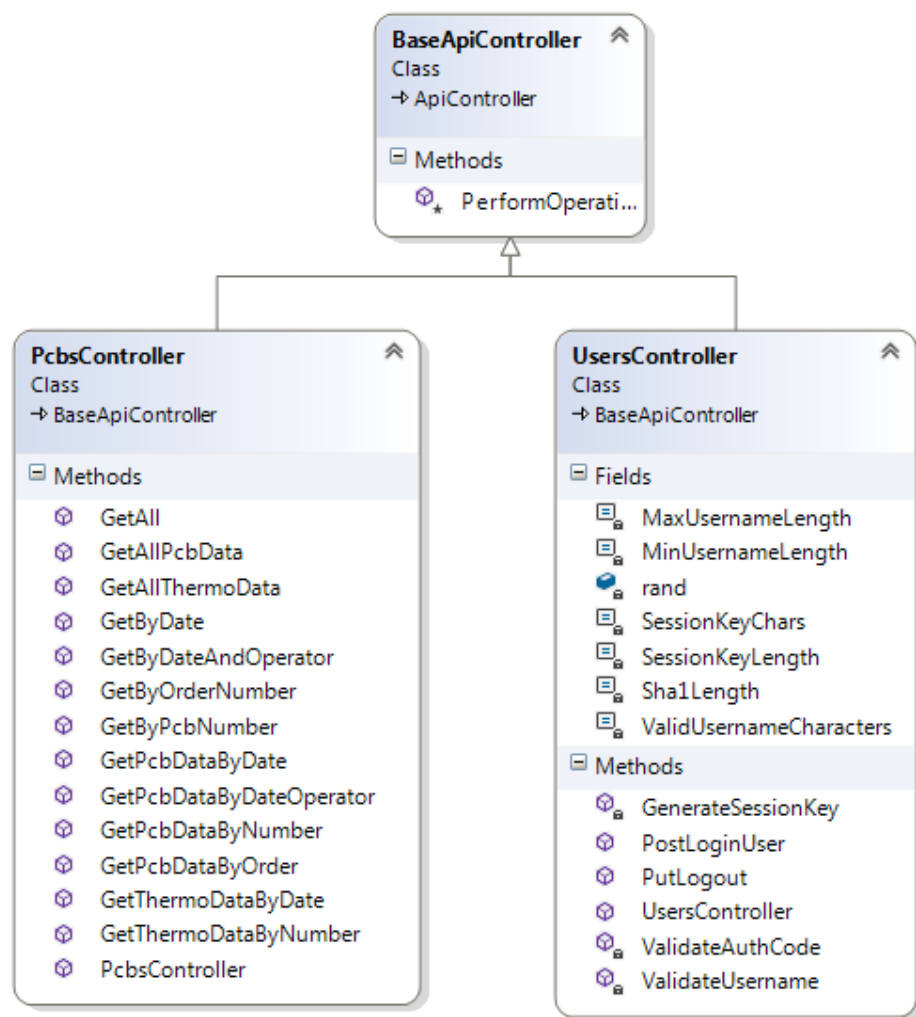
За да се осъществи извличането на информацията е необходимо да се реализира клиент-сървър архитектура. В настоящата глава ще бъдат изложени принципите за реализация на сървърното приложение.

Основни задачи на сървъра са: автентификация на потребителите в системата; връщане на информация по зададени критерии в отговор на запитвания от страна на клиентското приложение.

Сървърът представлява `ASP.NET Web API` приложение. Това е платформа за лесна разработка на `HTTP services` с голям брой поддържани формати. Избрана е тази форма на сървърно приложение заради възможността клиента, който ще „консумира“ сървисите да бъде десктоп, уеб или мобилен клиент с цел бъдещо развитие на системата.

### 5.5.1 Структура на приложението

Класовете на сървърното приложение са разделени на две групи: контролери и модели. Класовете на контролерите са показани на фигура 5.4



фигура 5.4 Диаграма на контролерите в приложението

### 5.5.2 Контролер BaseApiController

Създава се клас **BaseApiController**, който наследява ApiController, т.е контролерът по подразбиране. В този клас има само една функция, която се грижи за прихващане на грешките, възникнали по време на комуникацията със сървъра.

```

protected T PerformOperationAndHandleExceptions<T>(Func<T> operation)
{
    try
    {
        return operation();
    }
    catch (Exception ex)
    {
        var errResponse = this.Request.CreateErrorResponse(HttpStatusCode.BadRequest,
ex.Message);
        throw new HttpResponseMessage(errResponse);
    }
}
  
```

```
}
```

### 5.5.3 Контролер UsersController

Контролерът **UsersController** съдържа функциите за работа с потребители и потребителски данни. В него са реализирани следните методи:

- *private string GenerateSessionKey(int userId)* - Методът генерира сесийен ключ при вход на потребител в системата. Сесийния ключ служи за достъп до останалата функционалност на сървърното приложение и служи за удостоверяване правата на достъп на потребителя до тази информация
- *private void ValidateAuthCode(string authCode)* - Методът прави Валидация на подадения код за автентификация като при възникнала грешка връща съобщение за грешка
- *private void ValidateUsername(string username)* - Методът прави Валидация на подаденото потребителско име като при възникнала грешка връща съобщение за грешка

Контролерът реализира следните услуги:

- Услуга за вход в системата

Услугата за вход в системата се намира на:

```
http://services-root/users/login
```

Тази услуга позволява потребител да влезе в системата със съществуващ акаунт, като подава потребителско име и парола. Заявката има следната форма:

URL	http://services-root/users/login	Method: POST
Body	{ "username": "p_arbov", "authCode": "c79e32c27a1f1241f0da218d57bf15ea9e09e7dc" }	

Полето `authCode` представлява хешираната парола с помощта на алгоритъм SHA1. Тялото на заявката представлява обект от тип `UserModel`. Обектът се десериализира и се прави Валидация на полетата. Създава се контекст към базата данни. След това използвайки контекста, ако за конкретния потребител няма генериран сесийен ключ се генерира такъв.

```
if (user.sessionKey == null)
{
    user.sessionKey = this.GenerateSessionKey(user.id);
    context.SaveChanges();
}
```

При успех услугата връща HTTP код на статуса 200 и json обект в следната форма:

Body	{ "sessionKey": "172Jp3twchjt8I3DpI3Fdb4t5z7weEQgnDey5HZTDgJEBrp3Yn", "username": "p_arbov" }
------	--

Клиентското приложение, което ще използва услугите трябва да запазва върнатия сесийен ключ, понеже той е нужен за достъп до другите услуги на сървърното приложение

- Услуга за изход от системата

Услугата се намира на:

<code>http://services-root/user/logout</code>
---

Услугата прекратява текущата сесия на този потребител в системата, непозволявайки му достъп до услугите, освен тази за вход в системата.

<pre>if (user != null) {     user.sessionKey = null;     context.SaveChanges(); }</pre>
---

Потребителя може да влезе в системата отново, като получи нов сесийен ключ  
Заявката към услугата изисква на края на адреса на заявката да се добави сесийния ключ.

URL	<code>http://services-root/users/logout/{sessionKey}</code>	Method: PUT
-----	---	-------------

При успех услугата връща HTTP статус код 200

### 5.5.4 Контролер PcbController

Контролерът наследява създадения BaseApiController и реализира следните услуги:

- Услуга Get-all

Услугата се намира на адрес:

<code>http://services-root/user/scores</code>
---

Заявката към услугата изисква сесийен ключ да бъде добавен на края на адреса на заявката.  
Сесийния ключ се използва за автентификация на потребителя

URL	<code>http://services-root/user/scores/{sessionKey}</code>	Method: GET
Body	<code>(GET request =&gt; empty)</code>	

При успешна заявка услугата връща HTTP статус код 200 и json обект, който представлява списък от информация за платките, подредени по номер на платка. За формирането на формата на отговора се използва моделите PcbModel, PcbDataModel и PcbThermoModel.

<pre>var models =     (from pcbEntity in pcbEntities      select new PcbModel()      {          Id = pcbEntity.id,          PcbNumber = pcbEntity.pcbNumber,          Sensor = (from sensorEntity in pcbEntity.Sensors                   select sensorEntity.number),          PcbData = (from pcbDataEntity in pcbEntity.PCB_Data                    select new PcbDataModel()),          ThermoData = (from thermoDataEntity in pcbEntity.Thermo_Data                       select new PcbThermoModel())      }); return models.OrderByDescending(thr =&gt; thr.PcbNumber);</pre>
---

Отговорът има следния формат:



Body	<pre> {{   "Id":12,   "PcbNumber":1999,   "Sensor":[3394],   "PcbData":{{     "Id":12,"Date":"18-01-13 ",     "Time":"12:27:55 ",     "Operation":"Prog+EED",     "Model":"SL      ",     "Speed":"30-jet  ",     "BoxNumber":1999,     "Cal1":"Cow",     "Cal2":"Buffalo",     "Cal3":"Mix Milk",     "program":"Ls59-Ind-01-StartUp-FastCyc.hex",     "Data":"S-LS59-00-Steps4-StartUp.eed",     "ClientName":"5.9.2",     "OrderNumber":"M 39",     "Country":"India",     "Operator":"MH      ",     "ProgramVersion":7.1,     "Options":"NoOptions",     "PcbNumber":1999,     "SensorNumber":3394   }},   "ThermoData":{{     "Id":37,     "Date":"21-01-13",     "Time":"16:16:47",     "BoxNumber":1999,     "PcbSensor":3394,     "N40":1531,     "P60":2142,     "Workplace":"SNT:NK",     "ProgramVersion":1.1,     "Ad623":248,     "Amplitude":0,     "PcbNumber":1999}   }}   ...   {следващ елемент в списъка}   ... }} </pre>
------	---

В случай на грешка се връща HTTP код на грешка

- Услуга Get-by-pcb

Услугата се намира на адрес:

<http://services-root/pcbs/get-by-pcb>

Заявката към услугата изисква сесиен ключ и номерът за филтриране да бъдат добавени на края на адреса на заявката. Сесийният ключ се използва за автентификация на потребителя

URL	http://services-root/pcbs/get-by-pcb/{sessionKey}{searchNumber}	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки с номер, който е подаден в края на заявката. Отговорът се формира като заявката get-all се филтрира по подаден номер

```
var models = this.GetAll(sessionKey)
    .Where(pcb=>pcb.PcbNumber.Equals(searchNumber));
return models;
```

Отговорът има следния формат:

Body	<pre>{   "Id":12,   "PcbNumber":1999,   "Sensor":[3394],   "PcbData":{     "Id":12,"Date":"18-01-13 ",     "Time":"12:27:55 ",     "Operation":"Prog+EED",     "Model":"SL      ",     "Speed":"30-jet  ",     "BoxNumber":1999,     "Cal1":"Cow",     "Cal2":"Buffalo",     "Cal3":"Mix Milk",     "program":"Ls59-Ind-01-StartUp-FastCyc.hex",     "Data":"S-LS59-00-Steps4-StartUp.eed",     "ClientName":"5.9.2",     "OrderNumber":"M 39",     "Country":"India",     "Operator":"MH      ",     "ProgramVersion":7.1,     "Options":"NoOptions",     "PcbNumber":1999,     "SensorNumber":3394   },   "ThermoData":{     "Id":37,     "Date":"21-01-13",     "Time":"16:16:47",     "BoxNumber":1999,     "PcbSensor":3394,     "N40":1531,     "P60":2142,     "Workplace":"SNT:NK",     "ProgramVersion":1.1,     "Ad623":248,   } }</pre>
------	--

	<pre>"Amplitude":0, "PcbNumber":1999} }}</pre>
--	--

В случай на грешка се връща HTTP код на грешка

- Услуга Get-by-date

Услугата се намира на адрес:

<http://services-root/pcbs/get-by-date>

Заявката към услугата изисква сесиен ключ и дата за търсене да бъдат добавени на края на адреса на заявката. Сесийния ключ се използва за автентификация на потребителя

URL	<a href="http://services-root/pcbs/get-by-pcb/{sessionKey}{date}">http://services-root/pcbs/get-by-pcb/{sessionKey}{date}</a>	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки произведени на дата, която е подадена в края на заявката. Отговорът се формира като заявката get-all се филтрира по подадената дата

```
var models = this.GetAll(sessionKey)
    .Where(pcb => pcb.PcbData.Any(d => d.Date == date));
return models;
```

Отговорът има следния формат:

Body	<pre>{   "Id":12,   "PcbNumber":1999,   "Sensor":[3394],   "PcbData":[{     "Id":12,"Date":"18-01-13 ",     "Time":"12:27:55 ",     "Operation":"Prog+EED",     "Model":"SL      ",     "Speed":"30-jet  ",     "BoxNumber":1999,     "Cal1":"Cow",     "Cal2":"Buffalo",     "Cal3":"Mix Milk",     "program":"Ls59-Ind-01-StartUp-FastCyc.hex",     "Data":"S-LS59-00-Steps4-StartUp.eed",     "ClientName":"5.9.2",     "OrderNumber":"M 39",     "Country":"India",     "Operator":"MH      ",     "ProgramVersion":7.1,     "Options":"NoOptions",     "PcbNumber":1999,     "SensorNumber":3394   }], }</pre>
------	---

	<pre> "ThermoData":[{   "Id":37,   "Date":"21-01-13",   "Time":"16:16:47",   "BoxNumber":1999,   "PcbSensor":3394,   "N40":1531,   "P60":2142,   "Workplace":"SNT:NK",   "ProgramVersion":1.1,   "Ad623":248,   "Amplitude":0,   "PcbNumber":1999} }] ... {следващ елемент в списъка} ... ]</pre>
--	---

В случай на грешка се връща HTTP код на грешка

- Услуга **Get-by-date-operator**

Услугата се намира на адрес:

<http://services-root/pcbs/get-by-date-operator>

Заявката към услугата изисква сесиен ключ, дата и име на оператор да бъдат добавени на края на адреса на заявката. Сесийния ключ се използва за автентификация на потребителя

URL	<a href="http://services-root/pcbs/get-by-date-operator/{sessionKey}{date}{placeOperator}">http://services-root/pcbs/get-by-date-operator/{sessionKey}{date}{placeOperator}</a>	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки произведени на подадената дата от подадения оператор. Отговорът се формира като заявката get-all се филтрира по подадената дата и оператор

```

var models = this.GetAll(sessionkey)
    .Where(pcb => pcb.PcbData.Any(d => d.Date == date && d.Operator == placeOperator));
return models;
```

Отговорът има следния формат:

Body	<pre> [{   "Id":12,   "PcbNumber":1999,   "Sensor":[3394],   "PcbData":[{     "Id":12,"Date":"18-01-13 ",     "Time":"12:27:55 ",     "Operation":"Prog+EED",</pre>
------	---

	<pre> "Model": "SL      ", "Speed": "30-jet  ", "BoxNumber": 1999, "Cal1": "Cow", "Cal2": "Buffalo", "Cal3": "Mix Milk", "program": "Ls59-Ind-01-StartUp-FastCyc.hex", "Data": "S-LS59-00-Steps4-StartUp.eed", "ClientName": "5.9.2", "OrderNumber": "M 39", "Country": "India", "Operator": "MH      ", "ProgramVersion": 7.1, "Options": "NoOptions", "PcbNumber": 1999, "SensorNumber": 3394 }}, "ThermoData": [{   "Id": 37,   "Date": "21-01-13",   "Time": "16:16:47",   "BoxNumber": 1999,   "PcbSensor": 3394,   "N40": 1531,   "P60": 2142,   "Workplace": "SNT:NK",   "ProgramVersion": 1.1,   "Ad623": 248,   "Amplitude": 0,   "PcbNumber": 1999} ]} ... {следващ елемент в списъка} ... ]</pre>
--	--

В случай на грешка се връща HTTP код на грешка

- Услуга Get-by-order

Услугата се намира на адрес:

<http://services-root/pcbs/get-by-order>

Заявката към услугата изисква сесийен ключ и номер на заявката да бъдат добавени на края на адреса на заявката. Сесийния ключ се използва за автентификация на потребителя

URL	<a href="http://services-root/pcbs/get-by-pcb/{sessionKey}{orderNumber}">http://services-root/pcbs/get-by-pcb/{sessionKey}{orderNumber}</a>	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки с номер на поръчка, който е подаден в края на заявката. Отговорът се формира като заявката get-all се филтрира по подаден номер на поръчка

```
var models = this.GetAll(sessionKey)
    .Where(pcb => pcb.PcbData.Any(order => order.OrderNumber == orderNumber));
return models;
```

Отговорът има следния формат:

Body	<pre>{   "Id": 12,   "PcbNumber": 1999,   "Sensor": [3394],   "PcbData": {     "Id": 12, "Date": "18-01-13 ",     "Time": "12:27:55 ",     "Operation": "Prog+EED",     "Model": "SL ",     "Speed": "30-jet ",     "BoxNumber": 1999,     "Cal1": "Cow",     "Cal2": "Buffalo",     "Cal3": "Mix Milk",     "program": "Ls59-Ind-01-StartUp-FastCyc.hex",     "Data": "S-LS59-00-Steps4-StartUp.eed",     "ClientName": "5.9.2",     "OrderNumber": "M 39",     "Country": "India",     "Operator": "MH ",     "ProgramVersion": 7.1,     "Options": "NoOptions",     "PcbNumber": 1999,     "SensorNumber": 3394   },   "ThermoData": {     "Id": 37,     "Date": "21-01-13",     "Time": "16:16:47",     "BoxNumber": 1999,     "PcbSensor": 3394,     "N40": 1531,     "P60": 2142,     "Workplace": "SNT:NK",     "ProgramVersion": 1.1,     "Ad623": 248,     "Amplitude": 0,     "PcbNumber": 1999   } }, ... {следващ елемент в списъка} ... }</pre>
------	--

В случай на грешка се връща HTTP код на грешка

- Услуга Get-pcbdata

Услугата се намира на адрес:

<http://services-root/pcbs/get-pcbdata>

Заявката към услугата изисква сесиен ключ да бъде добавен на края на адреса на заявката. Сесийния ключ се използва за автентификация на потребителя

URL	<a href="http://services-root/pcbs/get-pcbdata/{sessionKey}">http://services-root/pcbs/get-pcbdata/{sessionKey}</a>	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки ,които имат данни в таблицата PCB\_Data от базата данни. За формирането на формата на отговора се използва модела PcbDataModel.

```
var pcbEntities = context.PCB_Data;
var models =
    (from pcbDataEntity in pcbEntities
     select new PcbDataModel()
    );
return models.OrderByDescending(thr => thr.PcbNumber);
```

Отговорът има следния формат:

Body	<pre>[{   "Id":12,   "Date":"18-01-13 ",   "Time":"12:27:55 ",   "Operation":"Prog+EED",   "Model":"SL      ",   "Speed":"30-jet  ",   "BoxNumber":1999,   "Cal1":"Cow",   "Cal2":"Buffalo",   "Cal3":"Mix Milk",   "program":"Ls59-Ind-01-StartUp-FastCyc.hex",   "Data":"S-LS59-00-Steps4-StartUp.eed",   "ClientName":"5.9.2",   "OrderNumber":"M 39",   "Country":"India",   "Operator":"MH      ",   "ProgramVersion":7.1,   "Options":"NoOptions",   "PcbNumber":1999,   "SensorNumber":3394} {следващ-елемент-от-списъка} ]</pre>
------	--

В случай на грешка се връща HTTP код на грешка

- Услуга Get-pcbdata-bynumber

Услугата се намира на адрес:

http://services-root/pcbs/get-pcbdata-bynumber
--

Заявката към услугата изисква сесийен ключ и номер на платка да бъдат добавени на края на адреса на заявката. Сесийният ключ се използва за автентификация на потребителя

URL	http://services-root/pcbs/get-pcbdata-bynumber/{sessionKey}{searchNumber}	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки с номер, който е подаден в края на заявката. Отговорът се формира като заявката get-pcbdata се филтрира по подаден номер

<pre>var models = this.GetAllPcbData(sessionKey)     .Where(pcb =&gt; pcb.PcbNumber.Equals(searchNumber)); return models;</pre>
---

Отговорът има следния формат:

Body	<pre>{   "Id":12,   "Date":"18-01-13 ",   "Time":"12:27:55 ",   "Operation":"Prog+EED",   "Model":"SL      ",   "Speed":"30-jet  ",   "BoxNumber":1999,   "Cal1":"Cow",   "Cal2":"Buffalo",   "Cal3":"Mix Milk",   "program":"Ls59-Ind-01-StartUp-FastCyc.hex",   "Data":"S-LS59-00-Steps4-StartUp.eed",   "ClientName":"5.9.2",   "OrderNumber":"M 39",   "Country":"India",   "Operator":"MH      ",   "ProgramVersion":7.1,   "Options":"NoOptions",   "PcbNumber":1999,   "SensorNumber":3394 }</pre>
------	---

В случай на грешка се връща HTTP код на грешка

- Услуга Get-pcbdata-bydate

Услугата се намира на адрес:

http://services-root/pcbs/get-pcbdata-bydate
--



Заявката към услугата изисква сесиен ключ и дата да бъдат добавени на края на адреса на заявката. Сесийният ключ се използва за автентификация на потребителя

URL	http://services-root/pcbs/get-pcbdata-bydate/{sessionKey}{date}	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки с номер, който е подаден в края на заявката. Отговорът се формира като заявката get-pcbdata се филтрира по подадена дата.

```
var models = this.GetAllPcbData(sessionKey)
    .Where(pcb => pcb.Date.Equals(date));
return models;
```

Отговорът има следния формат:

Body	<pre>{   "Id":12,   "Date":"18-01-13 ",   "Time":"12:27:55 ",   "Operation":"Prog+EED",   "Model":"SL      ",   "Speed":"30-jet  ",   "BoxNumber":1999,   "Cal1":"Cow",   "Cal2":"Buffalo",   "Cal3":"Mix Milk",   "program":"Ls59-Ind-01-StartUp-FastCyc.hex",   "Data":"S-LS59-00-Steps4-StartUp.eed",   "ClientName":"5.9.2",   "OrderNumber":"M 39",   "Country":"India",   "Operator":"MH      ",   "ProgramVersion":7.1,   "Options":"NoOptions",   "PcbNumber":1999,   "SensorNumber":3394} {следващ-елемент-от-списъка} ]</pre>
------	---

В случай на грешка се връща HTTP код на грешка

- Услуга Get-pcbdata-byorder

Услугата се намира на адрес:

http://services-root/pcbs/get-pcbdata-byorder

Заявката към услугата изисква сесиен ключ и номер на заявка да бъдат добавени на края на адреса на заявката. Сесийният ключ се използва за автентификация на потребителя

URL	http://services-root/pcbs/get-pcbdata-	Method: GET
-----	--	-------------

	byorder/{sessionKey}{orderNumber}	
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки с номер, който е подаден в края на заявката. Отговорът се формира като заявката get-pcbdata се филтрира по подаден номер на поръчка.

```
var models = this.GetAllPcbData(sessionKey)
    .Where(pcb => pcb.OrderNumber.Equals(orderNumber))
    .OrderBy(pcb => pcb.PcbNumber);
return models;
```

Отговорът има следния формат:

Body	<pre>[{   "Id":12,   "Date":"18-01-13 ",   "Time":"12:27:55 ",   "Operation":"Prog+EED",   "Model":"SL      ",   "Speed":"30-jet  ",   "BoxNumber":1999,   "Cal1":"Cow",   "Cal2":"Buffalo",   "Cal3":"Mix Milk",   "program":"Ls59-Ind-01-StartUp-FastCyc.hex",   "Data":"S-LS59-00-Steps4-StartUp.eed",   "ClientName":"5.9.2",   "OrderNumber":"M 39",   "Country":"India",   "Operator":"MH      ",   "ProgramVersion":7.1,   "Options":"NoOptions",   "PcbNumber":1999,   "SensorNumber":3394} {следващ-елемент-от-списъка} ]</pre>
------	--

В случай на грешка се връща HTTP код на грешка

- Услуга Get-thermodata

Услугата се намира на адрес:

http://services-root/pcbs/get-thermodata
--

Заявката към услугата изисква сесиен ключ да бъде добавен на края на адреса на заявката. Сесийния ключ се използва за автентификация на потребителя

URL	http://services-root/pcbs/get-thermodata/{sessionKey}	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки, които имат записани данни в таблица Thermo\_Data на базата данни. За формирането на формата на отговора се използва модела PcbThermoModel

```
var pcbEntities = context.Thermo_Data;
    var models =
        (from thermoDataEntity in pcbEntities
         select new PcbThermoModel()
         );
    return models.OrderByDescending(thr => thr.PcbNumber);
```

Отговорът има следния формат:

Body	<pre>[{   "Id":37,   "Date":"21-01-13",   "Time":"16:16:47",   "BoxNumber":1999,   "PcbSensor":3394,   "N40":1531,   "P60":2142,   "Workplace":"SNT:NK",   "ProgramVersion":1.1,   "Ad623":248,   "Amplitude":0,   "PcbNumber":1999 }]</pre> <p>{следващ-елемент-от-списъка}</p>
------	--

В случай на грешка се връща HTTP код на грешка

- Услуга Get-thermodata-bynumber

Услугата се намира на адрес:

<http://services-root/pcbs/get-thermodata-bynumber>

Заявката към услугата изисква сесийен ключ и номер на платка да бъдат добавени на края на адреса на заявката. Сесийния ключ се използва за автентификация на потребителя

URL	<a href="http://services-root/pcbs/get-thermodata-bynumber/{sessionKey}{searchNumber}">http://services-root/pcbs/get-thermodata-bynumber/{sessionKey}{searchNumber}</a>	Method: GET
Body	(GET request => empty)	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки с номер, който е подаден в края на заявката. Отговорът се формира като заявката get-thermodata се филтрира по подаден номер

```
var models = this.GetAllThermoData(sessionKey)
    .Where(pcb => pcb.PcbNumber.Equals(searchNumber));
return models;
```

Отговорът има следния формат:

Body	<pre>{   "Id":37,   "Date":"21-01-13",   "Time":"16:16:47",   "BoxNumber":1999,   "PcbSensor":3394,   "N40":1531,   "P60":2142,   "Workplace":"SNT:NK",   "ProgramVersion":1.1,   "Ad623":248,   "Amplitude":0,   "PcbNumber":1999 }</pre>
------	--

В случай на грешка се връща HTTP код на грешка

- Услуга Get-thermodata-bydate

Услугата се намира на адрес:

<http://services-root/pcbs/get-thermodata-bydate>

Заявката към услугата изисква сесиен ключ и дата да бъдат добавени на края на адреса на заявката. Сесийния ключ се използва за автентификация на потребителя

URL	<a href="http://services-root/pcbs/get-thermodata-bydate/{sessionKey}{date}">http://services-root/pcbs/get-thermodata-bydate/{sessionKey}{date}</a>	Method: GET
Body	<i>(GET request =&gt; empty)</i>	

При успешна заявка услугата връща HTTP статус код 200 и JSON обект, който представлява списък от платки с номер, който е подаден в края на заявката. Отговорът се формира като заявката get-thermodata се филтрира по подадена дата

```
var models = this.GetAllThermoData(sessionKey)
    .Where(pcb => pcb.Date.Equals(date));
return models;
```

Отговорът има следния формат:

Body	<pre>{   "Id":37,   "Date":"21-01-13",   "Time":"16:16:47",   "BoxNumber":1999,   "PcbSensor":3394,   "N40":1531,   "P60":2142,   "Workplace":"SNT:NK",   "ProgramVersion":1.1,   "Ad623":248, }</pre>
------	--

```

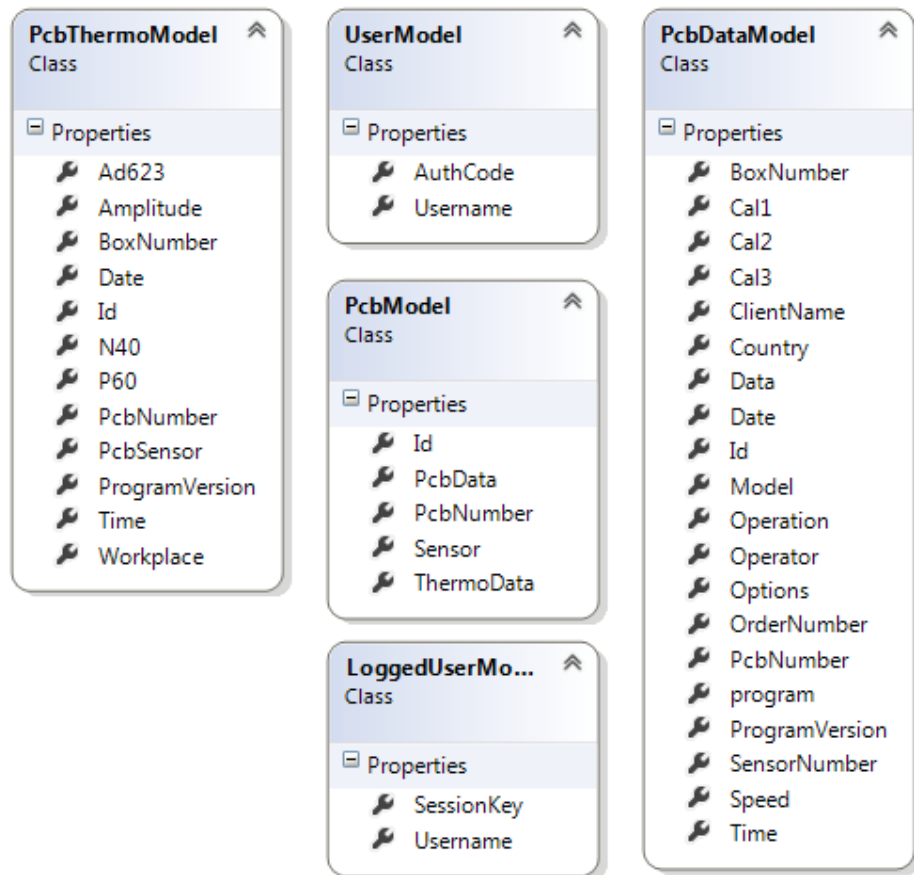
"Amplitude":0,
"PcbNumber":1999
}
{следващ-елемент-от-списъка}
]

```

В случай на грешка се връща HTTP код на грешка

### 5.5.5 Модели

Моделите са \*.cs класове, които служат за образуване формата на данните, които се изпращат като JSON обект в отговор на запитвания към услугите на сървърното приложение. Схема на моделите е показана на фигурата:



фигура 5.5 Диаграма на моделите в приложението

В описаните в глави 5.5.4 и 5.5.3 услуги на сървърното приложение, повечето от тях връщат обект или списък от обекти, генериран по един или комбинация от модели. Следния код описва създаването на обект от тип PcbThermoModel, който се връща в част от услугите на Контролера PcbController:

```

MilkotronicSystemEntities context = new MilkotronicSystemEntities();

var pcbEntities = context.Thermo_Data;
var models =
    (from thermoDataEntity in pcbEntities

```

```

select new PcbThermoModel()
{
    Id = thermoDataEntity.id,
    Date = thermoDataEntity.date,
    Time = thermoDataEntity.time,
    BoxNumber = thermoDataEntity.box,
    PcbSensor = thermoDataEntity.sensor,
    N40 = thermoDataEntity.N40,
    P60 = thermoDataEntity.P60,
    Workplace = thermoDataEntity.workplace,
    ProgramVersion = thermoDataEntity.programVersion,
    Ad623 = thermoDataEntity.ad623,
    Amplitude = thermoDataEntity.amplitude,
    PcbNumber = thermoDataEntity.numberPcb
});
return models.OrderByDescending(thr => thr.PcbNumber);

```

Всяко едно от полетата на модела се попълва с данни, извлечени от базата данни. След това се връща списъка обекти, генерирани по този модел и те се изпращат като отговор на заявката към конкретната услуга.

### 5.5.6 HTTP routes

HTTP routes са пътищата, създадени в сървърното приложение за достъп до услугите. Обикновено те имат структура *api/име-на-контролер/действие{списък-параметри}*. Пътищата се намират във файла WebApiConfig.cs в папката App\_Start на сървърното приложение от тип ASP .Net Web API.

За създадените контролери в приложението са създадени два нови HTTP маршрути:

- Маршрут за контролер UsersController

```

config.Routes.MapHttpRoute(
    name: "UsersApi",
    routeTemplate: "api/users/{action}",
    defaults: new
    {
        controller = "users"
    }
);

```

- Маршрут за контролер PcbController

```

config.Routes.MapHttpRoute(
    name: "PcbsApi",
    routeTemplate: "api/pcbs/{action}",
    defaults: new
    {
        controller="pcbs"
    }
);

```

Всеки нов път, който се създаде и е по-конкретен от пътя по подразбиране се записва преди този по подразбиране в конфигурационния файл.

## 5.6 Клиентско приложение

Клиентското приложение е стандартен windows изпълним файл. След стартирането му се появява форма, подканваща потребителя да въведе своето потребителско име и парола. Въведените данни се сериализират в json формат, изпращат се като http post заявка към сървъра и се прави опит за автентификация. При успех се показва основната форма на приложението. От тази форма чрез различните менюта се достига до останалата функционалност на приложението. В следващите точки ще бъдат разгледани подробно възможностите на потребителския интерфейс, който е изграден с помощта на windows forms. Всеки прозорец (или още форма) съдържа описание на външния си вид и описание на функционалността в \*.cs файл, написан на C#

### 5.6.1 Организация на приложението

След стартиране се показва първата форма на екрана (тази за автентификация). При успешен вход в система, от сървъра се връща сесийен ключ. Сесийният ключ е символен низ от 50 символа, който се предава като параметър във всички заявки към сървъра, освен “login” и служи за вътрешна аутентификация на потребителя. За да се запази този ключ във всички форми на приложението се създава Свойство (property), в което се запазва стойността на този ключ.

```
public string sessionKey { get; set; }
```

При преход от формата за вход към главната форма сесийния ключ се предава като параметър.

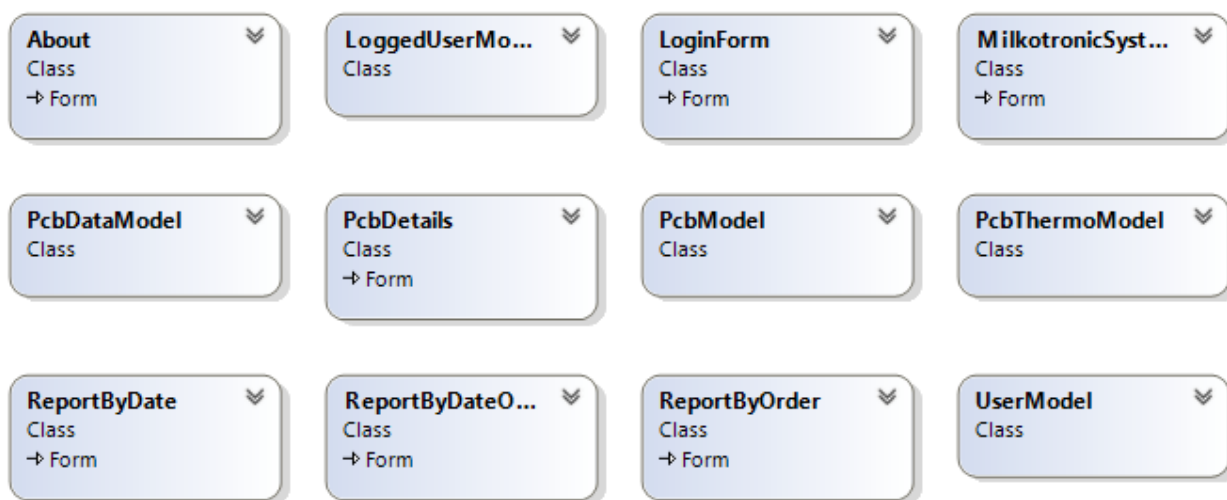
```
MilkotronicSystem mainform = new MilkotronicSystem(sessionKey);  
mainform.Show();
```

При успешен вход в системата се скрива формата за вход и се показва основната форма на приложението. В конструктора на формата се запазва стойността на сесийния ключ във свойството sessionKey. По този начин при преход от главната форма към друга, ключът ще може да бъде предаден като параметър.

От главната форма на приложението, чрез създадената лента за навигация може да се достигне до другите форми, изграждащи приложението, които ще бъдат разгледани подробно в следващите глави. При изход от системата се изпраща заявка “logout” към сървъра и сесийния ключ към конкретния потребител се нулира.

Създадените програмни файлове на приложението са два типа: файлова група от тип windows form и \*.cs (C# класове), в които са описани моделите, които се използват за сериализация и десериализация на данни по време на комуникацията със сървърното приложение.

Създадените форми са отделени в папка Views, а моделите в папка Models. Така се създава структура за по-лесна навигация в програмния код при нужда от промени. Обобщена схема на класовете и формите в приложението е показана на фигура 5.6



фигура 5.6 Диаграма на класовете в клиентското приложение

### 5.6.2 Хеширане на паролата

Потребителската парола никога не се изпраща в явен вид по мрежата, за да не може да бъде подслушана неправомерно. Вместо това се изпраща хеширана с криптографски алгоритъм SHA-1 репрезентация, което осигурява някакво ниво на защита. Библиотека от .Net Framework предоставя функции за такъв вид хеширане.

Методът има следния вид:

```
public static string CalculateSHA1(string text, Encoding enc)
{
    byte[] buffer = enc.GetBytes(text);
    SHA1CryptoServiceProvider cryptoTransformSHA1 = new SHA1CryptoServiceProvider();
    return BitConverter.ToString(cryptoTransformSHA1.ComputeHash(buffer)).Replace("-", "");
}
```

От подаден символен низ се изчислява символното шестнадесетично представяне на хеш кода. Класът BitConverter помага за конвертиране от двоичен формат в съответното символно представяне.

Паролата се хешира при вход в системата

### 5.6.3 Комуникация със сървър

За комуникация на клиентското приложение със сървърното приложение се използва класът WebClient. Класът WebClient е част от библиотеката System.Net. Методите на класът позволяват направата на синхронни заявки към сървър.

При комуникацията със сървър клиентското приложение осъществява 3 типа заявки:

- POST

POST заявка се изпраща при вход в системата. Заявката се осъществява с метода UploadString(url, data) на класа WebClient

```
var client = new WebClient();
client.Headers[HttpRequestHeader.ContentType] = "application/json";
```



```
var result = client.UploadString(userServiceUrl, userJson);
```

В атрибутите Headers се поставя типа данни, с които ще се извършва комуникацията. В приложението код се отправя заявка за вход в системата, при който към предварително съставения адрес на услугата в сървърното приложение се изпраща обект от тип UserModel, сериализиран в JSON формат. В променливата result се получава отговора на заявката, който при успешна заявка е обект от тип LoggedUserModel, получен в JSON формат.

- PUT

PUT заявка към сървърното приложение се изпраща при опит за изход в системата. За създаването на тази заявка се използва класът `HttpRequest`.

```
string url = "http://localhost:63494/api/users/logout?sessionKey=";  
url += sessionKey;  
  
var httpRequest = (HttpRequest)HttpRequest.Create(url);  
httpRequest.ContentType = "application/json";  
httpRequest.Method = "PUT";  
using (var streamWriter = new StreamWriter(httpRequest.GetRequestStream()))  
{  
    string json = "";  
  
    streamWriter.Write(json);  
}  
var httpResponse = (HttpWebResponse)httpRequest.GetResponse();
```

Към адреса на заявката за изход от системата се подава сесийния ключ

- GET

Всички останали заявки към сървърното приложение са от тип GET. Тези заявки не съдържат тяло на заявката. За осъществяването им е необходим само адреса на конкретната услуга от сървърното приложение

```
var client = new WebClient();  
var pcbs = client.DownloadString(url);
```

За осъществяването на заявката се използва метода `DownloadString` на класа `WebClient`

## 5.6.4 Генериране на доклади

Приложението реализира функционалност за генериране на доклади. Те представляват XML документи, в които се запазва подбрана информация за всяка платка по подадени различни критерии.

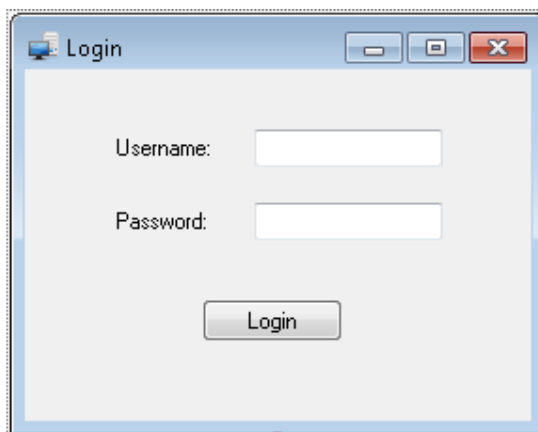
За създаването на документа се използват методите на класа `XmlDocument`.

За формат на документите е избран XML поради възможностите за преизползване им в бъдещо разширяване на функционалността на системата.

## 5.6.5 Прозорци на приложението

### 5.6.5.1 Вход в системата

При стартиране на клиентското приложение се появява прозорецът за вход в системата, който подканва потребителя да въведе своето потребителско име и парола.



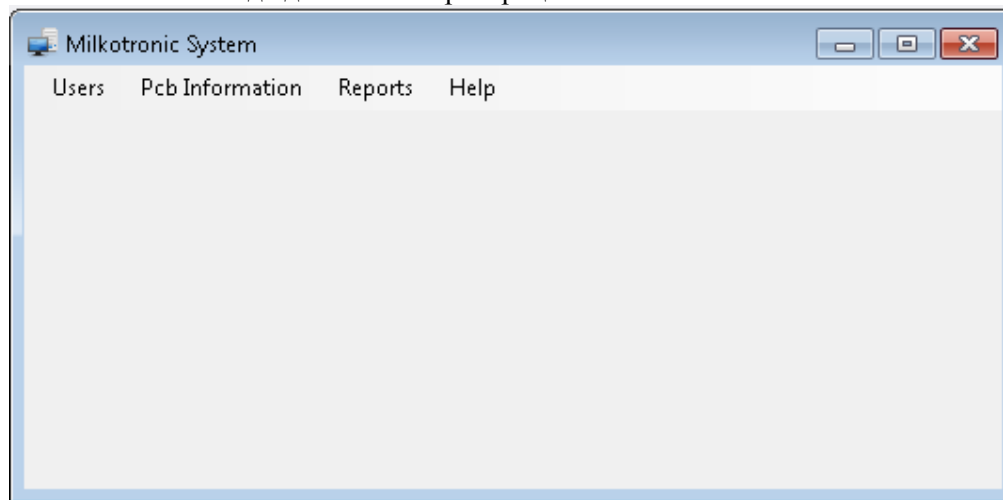
фигура 5.7 Изглед на прозорец Login

След натискане на бутона за вход, се прави опит за свързване със сървъра и изпращане на HTTP заявка към метода “login” на контролера “users”, като се изпращат като данни в заявката потребителското име и хешираната парола в json формат. За сериализацията на данните в този формат се използва библиотеката JSON.Net, която се добавя към проекта чрез мениджъра на пакети NuGet. Ако всичко е успешно формата за вход в системата се скрива и се показва основната форма на приложението. При грешка в някой от етапите се изкарва подходящо съобщение.

#### 5.6.5.2 Основен прозорец

При успешен вход в системата се показва главният прозорец на приложението. Основният прозорец служи за навигация към функционалностите, които реализира десктоп клиента. В менюто “Users” се намира бутон “Logout” чрез който се излиза от системата. Чрез менюто Pcb Information се достига до прозореца за визуализацията на цялата събрана информация за конкретна платка. В менюто “Reports” може да се избират критерии, по които да се създадат XML доклади.

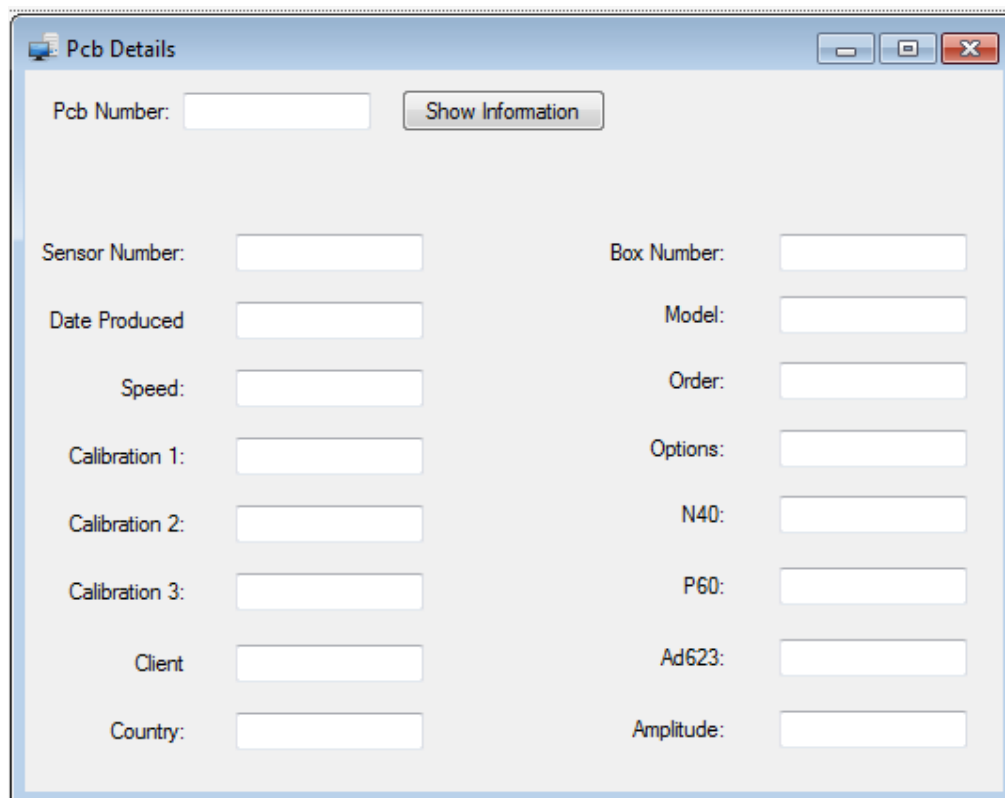
На фигурата е показан изгледа да главния прозорец:



фигура 5.8 Изглед на основния прозорец в приложението

#### 5.6.5.3 Прозорец за визуализиране на информацията за платка

Прозорецът за визуализиране на информацията е най-важния прозорец в приложението. Той се използва за визуализиране на цялата информация, която се пази в базата данни за конкретна платка. Изглед на прозореца е показан на фигурата:



фигура 5.9 Изглед на прозорец Pcb Details

При натискане на бутона се създава събитие, при което се изпраща GET заявка към get-by-pcb услугата на сървърното приложение. За осъществяването на тази заявка е необходим сесийния ключ, който се пази като свойство на инстанцията на формата и номер на платката, който се взима от текстовото поле на формата. С наличието на тези данни се формира адреса на заявката

```
string pcbNumber = tb_Search.Text;  
string url = "http://localhost:63494/api/pcbs/get-by-pcb?sessionKey=";  
url = url + sessionKey;  
url = url + "&searchNumber=";  
url = url + pcbNumber;
```

Следващата стъпка е да се направи заявката към сървърното приложение

```
var client = new WebClient();  
var pcbs = client.DownloadString(url);
```

След това се извършва десериализация на получените данни чрез модела PcbModel и се получава списък от платки с този номер.

```
IList<PcbModel> des = JsonConvert.DeserializeObject<IList<PcbModel>>(pcbs);
```

Понеже при попълването на базата данни се гарантира уникалността на записа, стойността на полето Count на получения списък е 1 при намерена платка и 0 при неналичие на резултат в базата данни. При намиране на съвпадение на съвпадение се взима първия елемент от списъка. След това се задават стойностите, които текстовите полета ще визуализират в прозореца

```

if (des.Count != 0)
{
    var desSensorData = des[0].Sensor;
    if (desSensorData.Count() != 0)
    {
        var sensorData = desSensorData.First();
        tb_Sensor.Text = sensorData.ToString();
    }

    var desPcbData = des[0].PcbData;
    if (desPcbData.Count() != 0)
    {
        var pcbData = desPcbData.First();
        tb_Cal1.Text = pcbData.Cal1;
        tb_Cal2.Text = pcbData.Cal2;
        tb_Cal3.Text = pcbData.Cal3;
        tb_Client.Text = pcbData.ClientName;
        tb_Country.Text = pcbData.Country;
        tb_Date.Text = pcbData.Date;
        tb_Model.Text = pcbData.Model;
        tb_Options.Text = pcbData.Options;
        tb_Order.Text = pcbData.OrderNumber;
        tb_Speed.Text = pcbData.Speed;
    }

    var desThermoData = des[0].ThermoData;
    if (desThermoData.Count() != 0)
    {
        var thermoData = desThermoData.First();
        tb_P60.Text = thermoData.P60.ToString();
        tb_N40.Text = thermoData.N40.ToString();
        tb_Ad623.Text = thermoData.Ad623.ToString();
        tb_Amplitude.Text = thermoData.Amplitude.ToString();
        tb_Box.Text = thermoData.BoxNumber.ToString();
        tb_Sensor.Text = thermoData.PcbSensor.ToString();
    }
}

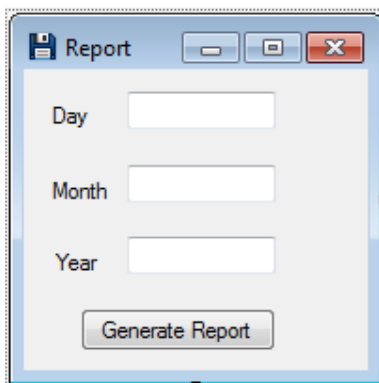
```

При успешна обработка на данните се извършва тяхната визуализация. При ненамиране на съвпадение на платка по номер в базата данни се изкарва съобщение, че не се намира платка с този номер

#### 5.6.5.4 Прозорец за генериране на доклад по дата

Този прозорец служи за въвеждане на дата, по която да се извърши филтриране на данните и да бъде генериран XML report с получените данни.

Изгледът на прозореца е показан на фигурата:



фигура 5.10 Изглед на прозорец за генериране на доклад по дата

След въвеждането на дата в текстовите полета и натискането на бутона се формира адреса за изпращане на заявката и се извършва самата заявка. Получените данни се десериализират до списък от обекти от тип `PcbDataModel`.

След това започва изграждането на XML документа

```
XmlDocument doc = new XmlDocument
XmlDeclaration dec = doc.CreateXmlDeclaration("1.0", null, null);
doc.AppendChild(dec); // Create the root element
XmlElement root = doc.CreateElement("report");
root.SetAttribute("date", date);
doc.AppendChild(root);
```

След това с оператор `foreach` се минава по списъка от елементи, като за всеки обект от списъка се създава дъщерен елемент в документа и се записват основни данни в атрибутите му

```
XmlElement analyzer = doc.CreateElement("analyzer");
XmlElement pcb = doc.CreateElement("pcb");
pcb.InnerText = item.PcbNumber.ToString();
XmlElement sensor = doc.CreateElement("sensor");
sensor.InnerText = item.SensorNumber.ToString();
XmlElement model = doc.CreateElement("model");
model.InnerText = item.Model;
XmlElement speed = doc.CreateElement("speed");
speed.InnerText = item.Speed;
XmlElement order = doc.CreateElement("order");
order.InnerText = item.OrderNumber;
XmlElement stationOperator = doc.CreateElement("operator");
stationOperator.InnerText = item.Operator;
analyzer.AppendChild(pcb);
analyzer.AppendChild(sensor);
analyzer.AppendChild(model);
analyzer.AppendChild(speed);
analyzer.AppendChild(order);
analyzer.AppendChild(stationOperator);
root.AppendChild(analyzer);
```

Последна стъпка в процеса е запазването на самия файл на твърдия диск на компютъра. В името на файла се добавя датата, за която е генериран.

На фигурата е показан примерен изглед на XML документа, генериран от функционалността на този прозорец.

```

<?xml version="1.0"?>
<report date="28-01-13">
  <analyzer>
    <pcb>2291</pcb>
    <sensor>387013</sensor>
    <model>SL    </model>
    <speed>30-jet </speed>
    <order>M 39</order>
    <operator>MH    </operator>
  </analyzer>
  ...

```

#### 5.6.5.5 Прозорец за генериране на доклад по дата и оператор

Този прозорец служи за въвеждане на дата и оператор, извършил записа на платката, по които да се извърши филтриране на данните и да бъде генериран XML report с получените данни. Изгледът на прозореца е показан на фигурата:

фигура 5.11 Изглед на прозорец за доклади по дата и оператор

След въвеждането на дата в текстовите полета и натискането на бутона се формира адреса за изпращане на заявката и се извършва самата заявка. Получените данни се десериализират до списък от обекти от тип PcbDataModel.

След това се следва логиката, описана в точка 5.6.5.4 с промени само в атрибутите на root елемента

```

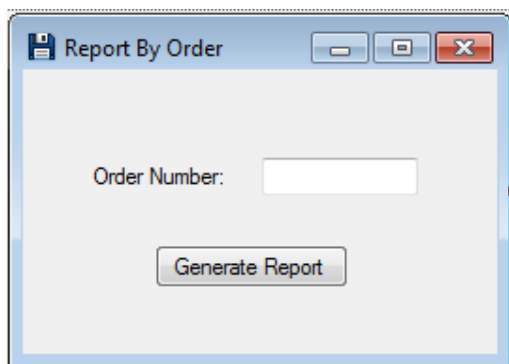
<?xml version="1.0"?>
<report date="28-01-13" operator="MH">
  <analyzer>
    <pcb>2291</pcb>
    <sensor>387013</sensor>
    <model>SL    </model>
    <speed>30-jet </speed>
    <order>M 39</order>
  </analyzer>
  ...

```

### 5.6.5.6 Прозорец за генериране на доклад по поръчка

Този прозорец служи за въвеждане на дата, по която да се извърши филтриране на данните и да бъде генериран XML report с получените данни.

Изгледът на прозореца е показан на фигурата:



фигура 5.12 Изглед на прозорец за генериране на доклад по поръчка

След въвеждането на дата в текстовите полета и натискането на бутона се формира адреса за изпращане на заявката и се извършва самата заявка. Получените данни се десериализират до списък от обекти от тип PcbDataModel.

След това се следва логиката, описана в точка 5.6.5.4 с промени само в атрибутите на root елемента

```
<?xml version="1.0"?>
<report order="M 39">
  <analyzer>
    <pcb>1869</pcb>
    <sensor>3390</sensor>
    <model>SL</model>
    <speed>30-jet</speed>
    <operator>MH</operator>
  </analyzer>
  ...

```

## 6. Ръководство за работа с приложението

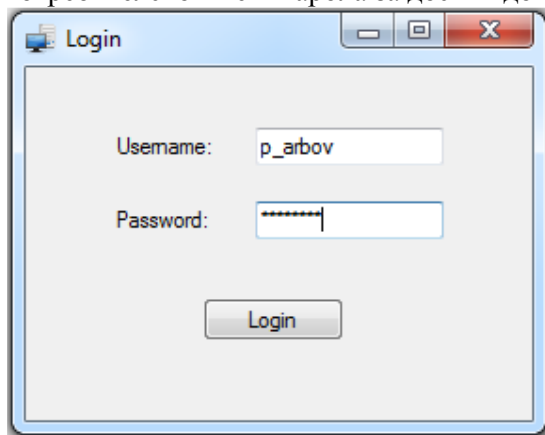
### 6.1 Внедряване на системата

Извършва се внедряване на системата в инфраструктурата, описана в точка 3. Това се извършва на следните стъпки:

- В релационната база данни SQL SERVER EXPRESS, която е инсталирана на сървъра (характеристиките са описани в точка 3) се създава нова база данни.
- В новосъздадената база данни се зарежда скрипта *MilkotronicSystem.Model.edmx.sql*, който е генериран от библиотеката за достъп до базата данни
- Стартира се приложението за попълване на данни в базата. За да се осигури автоматично обновяване и допълване на данни в базата се създава събитие чрез програмата Task Scheduler, която е вградена в Windows Server 2008, за стартирането на приложението всеки работен ден, след края на работното време
- Работните места, които ще използват данните, събирани в базата се намират в локалната мрежа на сървъра. Това дава възможност сървърното приложение да се стартира на компютърът сървър и да бъде достъпно от всеки компютър, член на мрежата
- Променят се зададените адреси на услугите в клиентското приложение, за да могат да достъпват новата локация на услугите
- Клиентското приложение е изпълним windows файл. За работа с него е необходимо да се стартира от компютър, който е член на мрежата на сървъра. Потребителят въвежда потребителското си име и паролата и има достъп до функционалността му

### 6.2 Работа с клиентското приложение

За работа с клиентското приложение е необходимо то да бъде стартирано от компютър, член на локалната мрежа на сървъра. След стартирането му се появява формата за вход в системата. Изисква се въвеждането на потребителско име и парола за достъп до системата.



фигура 6.1 Изглед на прозорец Login при изпълнение на приложението

След успешен вход в системата се показва основния прозорец на приложението. От там през менюто "Pcb Information" се достига до прозореца за визуализация на събраните данни.



The image shows a software window titled "Pcb Details". It contains a form with the following fields and values:

Field	Value
Pcb Number:	1999
Sensor Number:	3394
Box Number:	1999
Date Produced:	18-01-13
Model:	SL
Speed:	30jet
Order:	M 39
Calibration 1:	Cow
Options:	NoOptions
Calibration 2:	Buffalo
N40:	1531
Calibration 3:	Mix Milk
P60:	2142
Client:	5.9.2
Ad623:	248
Country:	India
Amplitude:	130

Фигура 6.2 Изглед на прозорец Pcb Details

Данните, визуализирани от този прозорец се използват в процеса на сервиз и поддръжка на устройствата.

През менюто "Reports" се избират критериите за създаване на доклад за производствения процес.

## **7. Заключение**

До тук бяха изложени методите за проектиране и реализация на системата. Бяха разгледани структурата на базата данни, начините за достъп до данните, структура и функционалност на приложението, което се грижи за попълване на данни в базата, сървърното и клиентското приложение. Основните предимства на представената система пред подобни софтуерни продукти са:

- Използва наличната инфраструктура
- Преносим сървър- той може да бъде стартиран в локалната мрежа, може да бъде качен и на безплатна клауд базирана платформа
- Клиентът е безплатен и не включва досадни реклами
- Клиентът е много бърз и използва малко ресурси
- Интерфейсът е изчистен и интуитивен

За подобряване на системата в бъдеще може да бъде добавена следната функционалност

- Създаване на уеб клиент
- Добавяне на функционалност за запазване информацията за направени поръчки
- Съхраняване и извличане осцилограмите на сигналите на сензорите в процеса на тестване
- Следене кои платки излизат от гаранция

Предвид направеното до тук, може да се заключи, че предметът на дипломната работа е един завършен и функциониращ софтуерен продукт.

## **8. Литература**

1. Наков, С. И др., Програмиране за .Net Framework, Faber, 2006
2. Наков, С. И колектив, Основи на програмирането със C#, 2012
3. Петцолд, Ч., Програмиране за Microsoft Windows на C#, Софтпрес, 2003
4. Guthrie, S., Using Linq to SQL, 2007
5. [http://en.wikipedia.org/wiki/Entity\\_Framework](http://en.wikipedia.org/wiki/Entity_Framework)
6. [http://en.wikipedia.org/wiki/Web\\_API#cite\\_note-1](http://en.wikipedia.org/wiki/Web_API#cite_note-1)