
**ФАКУЛТЕТ КОМПЮТЪРНИ СИСТЕМИ И ТЕХНОЛОГИИ**

**КОМПЮТЪРНО И СОФТУЕРНО ИНЖЕНЕРСТВО**
**РАЗРАБОТВАНЕ НА СОФТУЕР ЗА АВТОМОБИЛНАТА ИНДУСТРИЯ**

Разработване на потребителски графичен интерфейс (HMI) & Screen Streaming

# КУРСОВ ПРОЕКТ

**Образователно-квалификационна степен:** Магистър

**Разработили:**
Милен Вълчев
Радослав Нейков
Данило Христов
Петър Христов
Димитър Станишев

**Одобрил:** инж. Десислав Андреев

**София, 2020**

# Съдържание

## 1. Кратко описание на проекта

Разработва се софтуерно решение, което получава, записва и обработва данни от друг екип, който подава информация относно отчетените стойности от определени сензори (газов, звуков, ПИР, температурен и сензор за влажност).

За реализацията на backend логиката се използва Spring framework. Технологична рамка с отворен код за Java платформа. Тя предоставя много функции, които улесняват разработването на Java-базирани enterprise решение.

Пазим данните в релационната база данни с отворен код – PostgreSQL. Предимството й е, че поддържа множество типове данни и е мултиплатформена.

За изграждането на потребителския интерфейс си служим с Vue.js. Подходящ е за нашия случай, тъй като е събирателен за всички добри практики в различните JavaScript билблиотеки. Изключително гъвкав е и може да се използва в широк спектър от архитектурни решения. Допълнителен фактор е малкия размер, което води до по-бързо зареждане на уеб страниците. За бързодействието му се използва виртуален DOM (Document Object Model) за намиране на точното място за промяна, без да прегенерира целия реален DOM.

Общото между посочените технологии е, че са:

1. Силно разпространени.
2. Имат много добра документация.
3. Многото развити плъгини допринасят за по-лесната и приятна работа.

## 2. Backend програмна логика

Както по-горе описахме, за backend логиката използваме Spring Framework. На фиг. 1, имаме „мениджър“, който изважда и записва информация от базата данни.

```java
package com.tu.carinfocore.managers;

import com.tu.carinfocore.persistance.entities.MeteoInfoEntity;
import com.tu.carinfocore.persistance.entities.MeteoInfoEntityRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class MeteoInfoManager {

    @Autowired
    private MeteoInfoEntityRepository meteoInfoEntityRepository;

    public boolean saveMeteoInfo2DB(MeteoInfoEntity entity) {
        if (entity != null) {
            meteoInfoEntityRepository.save(entity);
            return true;
        }
        return false;
    }

    public Iterable<MeteoInfoEntity> getMeteoEntities() {
        return meteoInfoEntityRepository.findAll();
    }
}
```

*Фиг. 1.*

Решението се осъществява с помощта на два контролера. На фиг. 2 е показан контролер, който приема информация от другите системи посредством .json. На фиг. 3 е показан контролер, който комуникира с потребителския интерфейс.

```java
package com.tu.carinfocore.controllers;

import com.tu.carinfocore.managers.MeteoInfoManager;
import com.tu.carinfocore.persistance.dto.MeteoInfoDTO;
import com.tu.carinfocore.persistance.entities.MeteoInfoEntity;
import com.tu.carinfocore.utils.Converter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin
@RequestMapping(value = "/receive")
public class ReceiveController {

    @Autowired
    private MeteoInfoManager meteoManager;

    @RequestMapping(value = "/receiveMeteoInfo", consumes = MediaType.APPLICATION_JSON_VALUE,
    method = RequestMethod.POST)
    public boolean receiveMeteoInfo(@RequestBody MeteoInfoDTO dto) {
        MeteoInfoEntity entity = Converter.convertMeteoInfoDTO2MeteoInfoEntity(dto);
        return meteoManager.saveMeteoInfo2DB(entity);
    }
}
```

*Фиг. 2.*

```java
package com.tu.carinfocore.controllers;

import com.tu.carinfocore.managers.MeteoInfoManager;
import com.tu.carinfocore.persistance.dto.MeteoInfoDTO;
import com.tu.carinfocore.persistance.entities.MeteoInfoEntity;
import com.tu.carinfocore.utils.Converter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.MediaType;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;

@RestController
@CrossOrigin
@RequestMapping(value = "/api-ui")
public class RestUIController {

    @Autowired
    private MeteoInfoManager meteoManager;

    @RequestMapping(value = "/get-meteo-info", produces = MediaType.APPLICATION_JSON_VALUE,
    method = RequestMethod.GET)
    public List<MeteoInfoDTO> getMeteoinfo() {
        List<MeteoInfoDTO> dtos = new ArrayList<>();
        Iterable<MeteoInfoEntity> entities = meteoManager.getMeteoEntities();

        if (entities != null) {
            for (MeteoInfoEntity entity : entities) {
                MeteoInfoDTO dto = Converter.convertMeteoInfoEntity2MeteoInfoDTO(entity);
                dtos.add(dto);
            }
        }
        return dtos;
    }
}
```

*Фиг. 3.*

Приемаме модел от друга система – фиг. 4:

- entity записваме в базата
- dto – от интерфейса като го приемаме или изпращаме

```java
package com.tu.carinfocore.utils;

import com.tu.carinfocore.persistance.dto.MeteoInfoDTO;
import com.tu.carinfocore.persistance.entities.MeteoInfoEntity;

public class Converter {

    public static MeteoInfoEntity convertMeteoInfoDTO2MeteoInfoEntity(MeteoInfoDTO dto) {
        MeteoInfoEntity entity = new MeteoInfoEntity();
        entity.setTemperature(dto.getTemperature());
        entity.setHumidity(dto.getHumidity());
        entity.setEco2(dto.getEco2());
        entity.setTvoc(dto.getTvoc());
        entity.setDate(dto.getDate());
        entity.setHours(dto.getHours());
        return entity;
    }

    public static MeteoInfoDTO convertMeteoInfoEntity2MeteoInfoDTO(MeteoInfoEntity entity) {
        MeteoInfoDTO dto = new MeteoInfoDTO();
        dto.setTemperature(entity.getTemperature());
        dto.setHumidity(entity.getHumidity());
        dto.setEco2(entity.getEco2());
        dto.setTvoc(entity.getTvoc());
        dto.setDate(entity.getDate());
        dto.setHours(entity.getHours());
        return dto;
    }
}
```

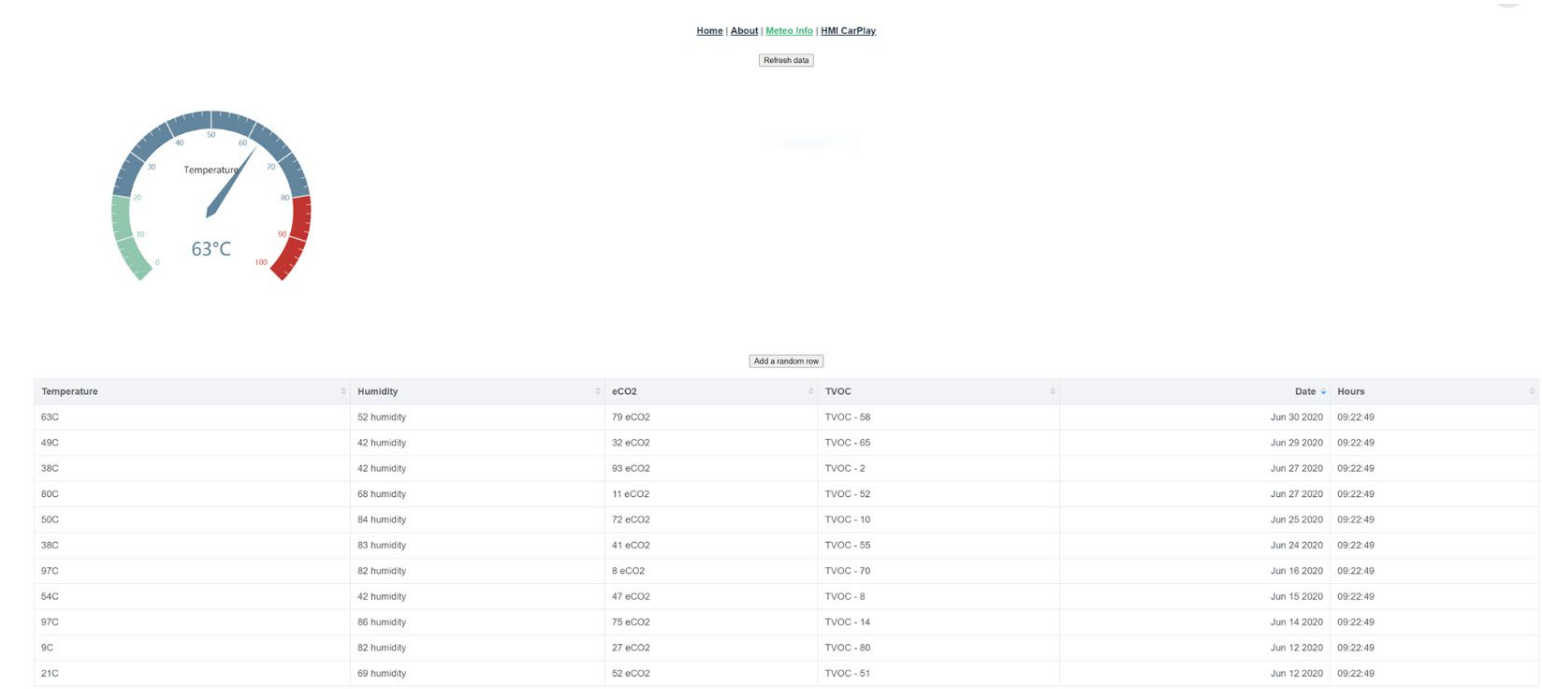
*Фиг. 4.*

## 3. Потребителски графичен интерфейс

Използваме Vue.js – бърза и лека JavaScript библиотека, която ни дава възможност да се обръщаме към backend логиката с помощта на плъгина Axios. Един от най-популярните Vue.js плъгини за REST повикване. Изключително лесен за използване, като има и добър error handling. Заявките, които правим към нашата логика са, за да вземем данните от метеорологичните сензори (информация, която се съхранява в базата) а след това тази информация визуализираме в таблица. На фиг. 5 може да се види как задаваме данни за таблиците и как четем от базата данни. След откъса от кода на страницата, може да се види как изглежда и самата страница на фиг. 6 (използваме TempGauge плъгин, за да визуализираме температурата в горния ляв ъгъл).

```
75      setTableData (rowsArray) {
76        rowsArray.sort(function (a, b) {
77          var keyA = new Date(a.Date)
78          var keyB = new Date(b.Date)
79          // Compare the 2 dates
80          if (keyA < keyB) return -1
81          if (keyA > keyB) return 1
82          return 0
83        })
84        this.$refs.customTable.rows = rowsArray
85      },
86      removeRow (index) {
87        this.$refs.customTable.rows.splice(index, 1)
88      },
89      loadDbData () {
90        axios.get('http://localhost:8080/car-info-core/api-ui/get-meteo-info')
91          .then(response => {
92          // JSON responses are automatically parsed.
93            this.setTableData(response.data)
94          })
95          .catch(e => {
96            this.errors.push(e)
97          })
98      }
```

*Фиг. 5.*


Home | About | Meteo Info | HMI CarPlay
Refresh data
Temperature
63°C
Add a random row
Temperature
Humidity
eCO2
TVOC
Date
Hours
63C
52 humidity
79 eCO2
TVOC - 58
Jun 30 2020
09:22:49
49C
42 humidity
32 eCO2
TVOC - 65
Jun 29 2020
09:22:49
38C
42 humidity
93 eCO2
TVOC - 2
Jun 27 2020
09:22:49
80C
68 humidity
11 eCO2
TVOC - 52
Jun 27 2020
09:22:49
50C
84 humidity
72 eCO2
TVOC - 10
Jun 25 2020
09:22:49
38C
83 humidity
41 eCO2
TVOC - 55
Jun 24 2020
09:22:49
97C
82 humidity
8 eCO2
TVOC - 70
Jun 16 2020
09:22:49
54C
42 humidity
47 eCO2
TVOC - 8
Jun 15 2020
09:22:49
97C
86 humidity
75 eCO2
TVOC - 14
Jun 14 2020
09:22:49
9C
82 humidity
27 eCO2
TVOC - 80
Jun 12 2020
09:22:49
21C
69 humidity
52 eCO2
TVOC - 51
Jun 12 2020
09:22:49

*Фиг. 6.*

## 4. Screen Streaming – Проучване на софтуерния проблем

Софтуерният проблем, за който трябва да намерим решение, а именно – видео предаването на целия или част от екрана на дадено устройство, в случая на проекта това е друго приложение, което измерва текущата скорост на автомобил в движение, и визуализацията му върху друг екран би могло да се раздели на две основни софтуерни задачи за изпълнение, а това са именно:

- ✓ създаването на поток от видео данни – видео фреймове, които да обхващат необходимата зона от екрана
- ✓ измислянето на начин за предаване на събирания видео поток на екрана на друго устройство

Изпълнението на първата задача, независимо от операционната система е свързано с определени права (permissions), които потребителят е необходимо да разреши с цел създаването на поток от видео данни, като източникът на тях е екранът на клиентското устройство. Вариантите за създаването на подобен видео поток главно можем да разделим на два, а именно:

- локален видео поток, имплементиран, като функционалност в приложението, от което искаме да вземем дадените видео данни
- глобално видео споделяне, като отделно (third party) приложение, имащо възможността да подава всичко, което се случва на екрана

За реализацията на втората задача – подаването на събираната видео информация от екрана на устройството, съответно е необходимо да сме готови с реализацията на първата, тъй като двете са взаимно зависими.

Основните варианти, които бихме могли да разгледаме за решаване на тази софтуерна задача, зависят главно от това къде се намира устройството, което трябва да получи и визуализира видео потока:

- ако двете устройства се намират непосредствено близо едно до друго – те биха могли да комуникират чрез локалната за тях мрежа
- при положение, че двете устройства са отдалечени или тяхната локация е различна – най-удачно би било използването на публична мрежа, през която да се случва комуникацията

След детайлното разделяне на софтуерния проблем на няколко отделни подзадачи с различни варианти за реализацията им е необходимо да изберем най-подходящото софтуерно решение за реализацията му.

## 4.1. Избор на софтуерно решение

За софтуерната реализация на задачата направихме собствено проучване за това колко време би отнело разработването на персонализирано за проекта решение от нулата, както и какви вече-съществуващи (third party) варианти има и дали някое от тях би било подходящо да използваме за целите на заданието.

За успешната реализация на проекта, взехме под внимание зависимостта от клиентското приложение, от което ще взимаме потока от видео данни. Именно заради това разгледахме множество варианти за цялостна разработка за създаването на видео потока и в последствие предаването му към отделна услуга, която роля е да го разпространи.

Проучването ни показа, че вариантът за цялостна разработка обаче крие редица зависимости, които трябва да вземем предвид, ако се спрем над него, а част от тях са именно:

- операционната система на клиентското устройство, за което самото приложение, от което взимаме видео потока бива разработвано
- езикът на програмиране и съответно софтуерната рамка (framework), която екипът използва за реализацията на приложението
- библиотеките, които са налични за съответната операционна система, в комбинация с използвания програмен език
- правата за достъп, които трябва да бъдат имплементирани още при изграждането (build) на клиентското приложение, които са различни за всяка операционна система
- технологичното време за разработка, което би ни отнело създаването на подобно решение от нулата

Изброените по-горе зависимости ни накараха да разгледаме и в крайна сметка да се спрем на варианта, в който използваме вече-съществуващо решение и го имплементираме за целите на проекта. Оказа се, че съществуват редица вече готови услуги за запис на даден екран, но малка част от тях предлагат излъчваният видео поток да бъде предаван в реално време на екрана на друго устройство.

След редица тестове успяхме да намерим две софтуерни решения, отговарящи на нашите изисквания, както и значително намаляващи зависимостите спрямо варианта с цялостна разработка . Едната услуга дава възможност за споделяне на видео поток от данни между две устройства в една и съща мрежа посредством рутер, а другата дори премахва този лимит, както и това устройствата да се намират непосредствено близо

едно до друго, като предава информацията на сървър – посредник, имащ за цел да препрати данните в реално време на другата от двете страни.

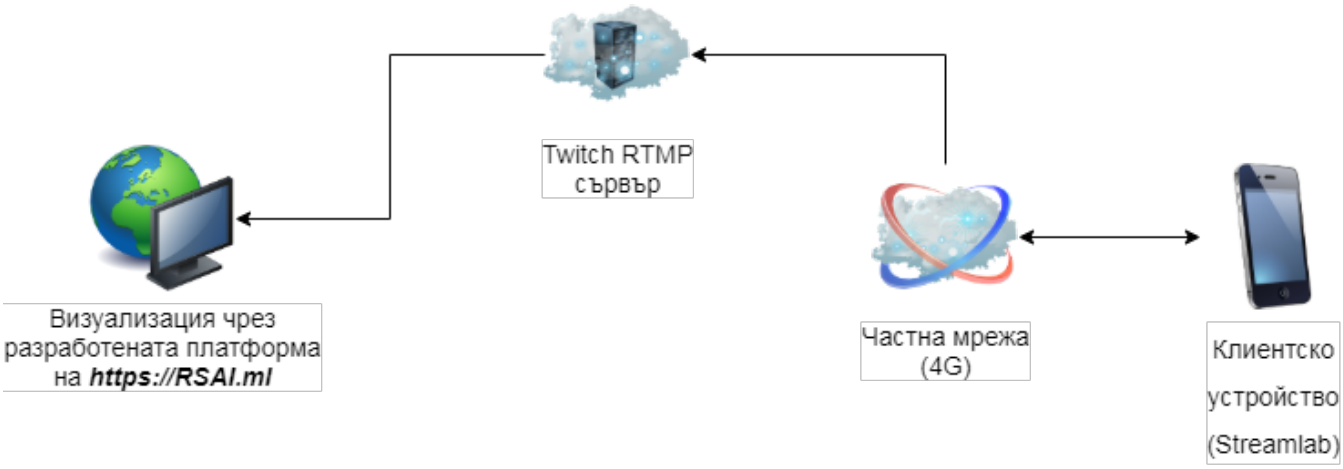
## 4.2. Техническа реализация на Streaming функционалността

За техническата реализация на разработвания проект, избрахме да използваме гореспоменатите две услуги, достъпни за най-разпространените мобилни операционни системи в днешно време – iOS и Android. Това ни предостави възможността да не сме зависими, нито от средата за разработка на клиентското приложение за измерване на скоростта, нито от програмния език, софтуерната рамка и библиотеките, налични конкретно за него.

### 4.2.1. Streamlab и RTMP сървър

Streamlab представлява услуга, предоставяща едновременно възможността за извличане на видео поток от данни от целия екран или част от него на дадено устройство в реално време и препращането им към RTMP (Real-Time Messaging Protocol) сървър, който се грижи за предаването им нататък. За реализацията на проекта решихме да използваме безплатна RTMP услуга, като се спряхме на Twitch. Причината за това е, че ресурсите, необходими за поддръжката на собствена такава не са малки, предвид факта, че подавания поток от данни е във видео формат.

Именно заради това интегрирахме в Streamlab услугата – Twitch под формата на сървър, предаващ видеото към изходното ни устройство – в случая това е web сървър, на който разработихме платформа за визуализация на получаваното съдържание в реално време, както е показано на фиг. 1. Всеки потребител с линк към домейна би могъл да гледа в реално време излъчвания видео поток, като това може да бъде направено на специално разработената платформа, показана на фиг. 2 на следния адрес: [https://RSAI.ml](https://RSAI.ml)




*Фиг. 1. Модел на предаване на видео потока от данни между потребителското устройство посредством частна (4G) мрежа, подаваща видео поток към Twitch RTMP сървъра и съответно устройство, визуализиращо информацията чрез изградената онлайн платформа чрез достъп до Интернет.*


*Фиг. 2. Потребителски интерфейс на разработената streaming платформа за проекта на [https://RSAI.ml](https://RSAI.ml)*

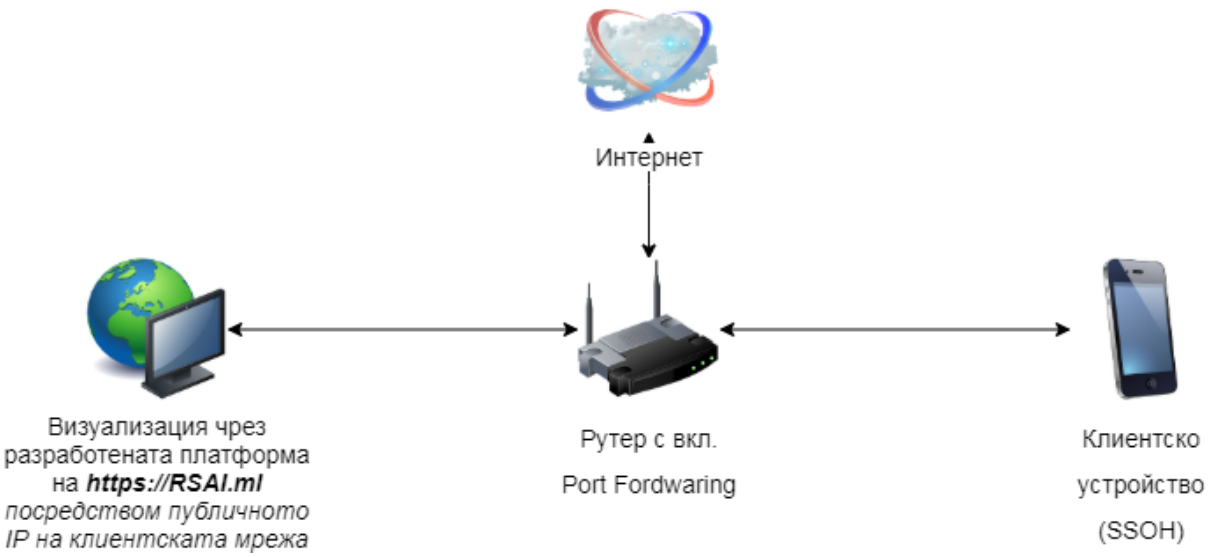
### 4.2.2. Споделяне на екрана посредством HTTP

Другата услуга, която решихме да използваме за реализация на предаването на видео поток в локална мрежа между две устройства, което би могло да бъде реализирано на практика безпрепятствено в един автомобил, се нарича Screen Stream over HTTP (SSOH). Чрез нея потребителят, както при Streamlab, има избора дали да сподели целия екран или да започне видео предаване на конкретна зона, предварително дефинирана чрез координати в мерната единица пиксел.

Това, по което се различава е, че е необходимо потребителското устройство да е свързано към локална мрежа, която притежава статично IP – за разлика от горния метод, който това може да бъде направено и чрез 4G мобилна мрежа с динамичен IP адрес. След това е необходимо рутерът, към който устройството е свързано да има отворен конкретен порт, чрез който видео връзката да може да бъде достъпена публично от външна мрежа (Port Forwarding), както е показано на фиг. 3. По този начин

видео потокът отново може да бъде визуализиран в разработената платформа, посредством следния шаблон:

*публичният IP адрес на мрежата : портът, който услугата използва*




*Фиг. 3. Модел на предаване на видео потока от данни между потребителското устройство посредством рутер и изградената онлайн платформа – чрез публичното IP на мрежата и съответния порт, на който рутерът е настроен да подава видео данните.*

### 4.3. Практическо изследване на постигнатите резултати

След подготовката за техническата реализация на проекта, се погрижихме за финалното тестване съвместно с екипа, разработващ клиентското приложение за измерване на скоростта. От наша страна бяха подадени инструкции за лесното имплементиране на услугите, както и стъпките, необходими от тяхна страна за демонстрирането на двата работещи screen streaming варианта.

Демонстрацията проведохме на разработената уеб платформа, специално за целите на проекта на https://RSAI.ml, като бяха тествани и двата реализирани варианта – в мобилна мрежа с динамичен IP адрес и в локална мрежа със статично IP и рутер. Тестовете показаха безпроблемната работа, както на едната, така и на другата софтуерна реализация и ясно показаха постигнатите резултати.

### 4.4. Заключение

Вярваме, че избрахме възможно най-добрите варианти за реализацията на софтуерното решение, взимайки предвид системните изисквания и времето, с което разполагаме за разработването на проекта. В заключение, поставеният софтуерен проблем за реализация на услуга, която предава видео поток на целия или част от екрана на дадено устройство, като след това го визуализира върху друг, намери не едно, а цели две работещи и на практика приложим решения, които биха могли да се използват при разработката на софтуер за автомобилната индустрия.