

POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY



Projekt zespołowy

Infoboard

<https://github.com/hobitolog/InfoBoard>

Krystian Minta 126800 - krystian.minta@student.put.poznan.pl
Robert Kosakowski 126813 - robert.kosakowski@student.put.poznan.pl
Maciej Jaśkiewicz 126825 - maciej.jaskiewicz@student.put.poznan.pl

Prowadzący:
mgr inż. Przemysław Walkowiak

Poznań, 2018

Uzasadnienie wyboru tematu	3
Założenia projektu	5
Podział prac pomiędzy członków zespołu	6
Funkcjonalność aplikacji	7
Wybrane technologie	9
Narzędzia, środowisko, biblioteki	11
Architektura rozwiązania	12
Interesujące problemy i rozwiązania ich na jakie się natknęliśmy	13
Opis zapytań sieciowych	14
Instrukcja użytkowania aplikacji	19
Wyświetlane widoki	26
Możliwości rozwoju	28
Podsumowanie	29
Źródła wiedzy	30

1. Uzasadnienie wyboru tematu

Temat projektu wybraliśmy podczas dyskusji. Interesowały nas przede wszystkim tematy z kategorii “różne”:

- Konferencje audio-video.
- Tablica Informacyjna.
- Podłączenie dodatkowego ekranu przez sieć IP.

Za pierwszym tematem przemawiało bardzo wiele zalet. Bardzo ważne jest zapotrzebowanie rynku na podobne systemy. Co mogłoby się przekształcić w przyszłe korzyści finansowe. Wiele firm byłoby zainteresowanych systemem pozwalającym na prowadzenie wysokiej jakości konferencji i szkoleń, bez instalacji dodatkowego oprogramowania na komputerze, ani profesjonalnego sprzętu. Aplikacja webowa to idealne rozwiązanie z powodu wieloplatformowości. Skorzystalibyśmy z nowoczesnych technologii jak HTML5 i websockets, które użyliśmy również w innych projektach. Jednak kwestie przechwytywania dźwięku i obrazu z wielu źródeł, miksowanie ich, zakodowanie oraz transmisja mogły sprawić nam wiele problemów. Przede wszystkim pod względem optymalizacyjnym. A wydajność aplikacji związanych z przetwarzaniem obrazu jest priorytetem.

Drugi projekt wydał nam się ciekawy z powodu wykorzystania **Single Board Computer** takiego jak na przykład Raspberry Pi, które jest cały czas rozwijane i posiada najróżniejsze projekty stworzone przez społeczność. Daje to wiele inspiracji i mobilizacji do stworzenia własnego oprogramowania na tę platformę. Na dodatek codziennie przechodząc po kampusie Politechniki Poznańskiej widzimy telewizory pokazujące ostatnie oraz nadchodzące wydarzenia, które mogą zainteresować studentów. Znow w rozważeniu komercyjnej aplikacji istnieje możliwość znalezienia kupca na taką aplikację. W tym wypadku po raz kolejny niezmiernie ważna jest wydajność. Nie dość, że aplikacja pracuje na systemie o nieporównywalnie niższej wydajności w porównaniu do maszyn na których codziennie pracujemy to założeniem projektu jest, iż w pewnych wypadkach minikomputer musi odtwarzać filmy, bądź transmisję z Internetu. Co oznacza, że nasza aplikacja musi wymagać minimalnej mocy obliczeniowej, by zostawić zapas dla kodeków obrazu.

Trzeci projekt wzbudził nasze zainteresowanie, ponieważ jako informatycy preferujemy pracę na wielu monitorach. Przykładowo w ustawieniu monitor laptopa na środku. Po prawej ekran telewizora o rozdzielczości 4K podłączony do laptopa za pomocą kabla HDMI. Telewizor posiada proporcje 16:9, czyli znajduje się w orientacji poziomej. Z kolei po lewej stronie laptopa jest monitorem Full HD 9:16, czyli w orientacji pionowej. Jednak laptopy w przeciwieństwie do komputerów stacjonarnych posiadających znacznie mocniejsze komponenty, takie jak karty graficzne z wieloma złączami HDMI (High Definition Multimedia Interface), DisplayPort, VGA (Video Graphics Array), czy mini DisplayPort, nie posiadają wielu złącz do podłączenia monitorów zewnętrznych. Zazwyczaj jest to pojedynczy port wykorzystany w naszym wypadku do podłączenia telewizora. Cóż zrobić w takim wypadku? Jednym z rozwiązań są zewnętrzne układy graficzne. Jednak ich cena oscyluje w bardzo wysokich przedziałach cenowych. Rozwiązaniem tego problemu mogłoby być wykorzystanie starego komputera stacjonarnego, którego używaliśmy przed zakupem nowego laptopa gamingowego. Stary blaszak zamiast kurzyć się w piwnicy może nam posłużyć do wyświetlania treści na naszym trzecim monitorze. Jednak taka aplikacja wymaga doskonałej znajomości technologii sieciowych między innymi takich jak protokoły IP (Internet Protocol), TCP (Transmission Control Protocol), UDP (User Datagram Protocol). Niezbędne jest też przetwarzanie obrazu, by grafika na dodatkowym monitorze była płynna. Ograniczeniem, które zdyskwalifikowało ten projekt jest różnorodne podejście do wyświetlania obrazu na różnych systemach operacyjnych. W naszym przypadku kiedy chcemy zmiennie pracować na różnych systemach operacyjnych takich jak Microsoft Windows 10, Linux Ubuntu, czy macOS High Sierra musielibyśmy uruchamiać trzy osobne aplikacje. Inne rozwiązania wymagałyby znacznie większych nakładów czasów niż przewiduje siatka godzin ECTS (European Credit Transfer System).

Ostateczną decyzję podjeliśmy korzystając z faktu, iż dwóch z trzech autorów projektu jest członkami koła naukowego Telewizja studencka Politechniki Poznańskiej SpacjaTV. Jednogłośnie stwierdziliśmy, że używamy profesjonalnego telewizyjnego sprzętu, więc zrezygnowanie z niego nie jest wskazane. Z kolei projekt tablicy informacyjnej może nam pomóc rozwinąć działalność koła naukowego. Dzięki tej aplikacji możemy konkurować z działem promocji Politechniki Poznańskiej.

Razem zdecydowaliśmy, że wybieramy projekt tablicy interaktywnej, a także nazwę naszego projektu - InfoBoard.

2. Założenia projektu

Wymagania stawiane przed projektem koncentrowały się na zapewnieniu niezawodności i wydajności systemu.

- Funkcjonalne
 - System logowania.
 - Podgląd wyświetlanego ekranu w przeglądarce.
 - Podgląd kolejki wydarzeń.
 - Dodawanie, edycja oraz usuwanie wydarzeń.
 - Tworzenie harmonogramu wyświetlania.
 - Wyświetlanie obrazów zgodnie z harmonogramem.
 - Wyświetlanie filmów zgodnie z harmonogramem.
 - Wyświetlanie stron www zgodnie z harmonogramem.
 - Kopia stron www do wyświetlania offline.
 - Wyświetlanie filmów na platformie YouTube zgodnie z harmonogramem.
 - Wyświetlanie transmisji na żywo na platformie YouTube zgodnie z harmonogramem.
 - Wyświetlanie paska wiadomości zgodnie z harmonogramem.
 - Wyświetlanie aktualnego czasu zgodnie z harmonogramem.
 - Zapis dodawanych plików w pamięci lokalnej.
- Niefunkcjonalne
 - Zdalny dostęp do panelu administracyjnego.
 - Wyświetlanie filmów w rozdzielczości Full HD.
 - Wyświetlanie transmisji w rozdzielczości Full HD.
 - Aplikacja przystosowana do SBC.
 - Pracuje na systemie Raspbian.
 - Panel administracyjny dostępny z dowolnej przeglądarki internetowej.
 - Aplikacja napisana w języku JavaScript.
 - Dane przechowywane w pliku JSON.

3. Podział prac pomiędzy członków zespołu

Prace nad projektem podzieliliśmy uwzględniając nasze doświadczenie przy innych projektach.

Robert jest administratorem swojego własnego serwera publicznego, na którym udostępnia różne aplikacje w architekturze klient-serwer bazujące na środowisku node.js. Dlatego Jego głównym zadaniem było przygotowanie serwera całej aplikacji. Wykonał trzecią niepublikowaną prezentację, która podsumowała nasze postępy.

Maciek stworzył system inteligentnego domu, z którego korzysta u siebie w mieszkaniu. System ten bazuje na platformie Raspberry Pi, dlatego On zajął się wdrożeniem aplikacji na SBC. Ponadto w innym projekcie zajmuje się front-endem. Z tego powodu stworzył moduł odpowiadający za wyświetlanie mediów na ekranie poprzez electron.js. Wykonał pierwszą prezentację w Prezi.

Krystian specjalizuje się w bazach danych oraz większych projektach korzystających z olbrzymich frameworków takich jak Spring i Hibernate. To też Jego umiejętności dały mu możliwość przygotowania struktury pliku JSON, będącego bazą danych wszystkich wyświetlanych na tablicy materiałów. Dodatkowo wdrożył On również obsługę zdarzeń CRON, które decydują o aktualnie wyświetlanym materiale. Wykonał drugą prezentację za pomocą Prezentacji Google.

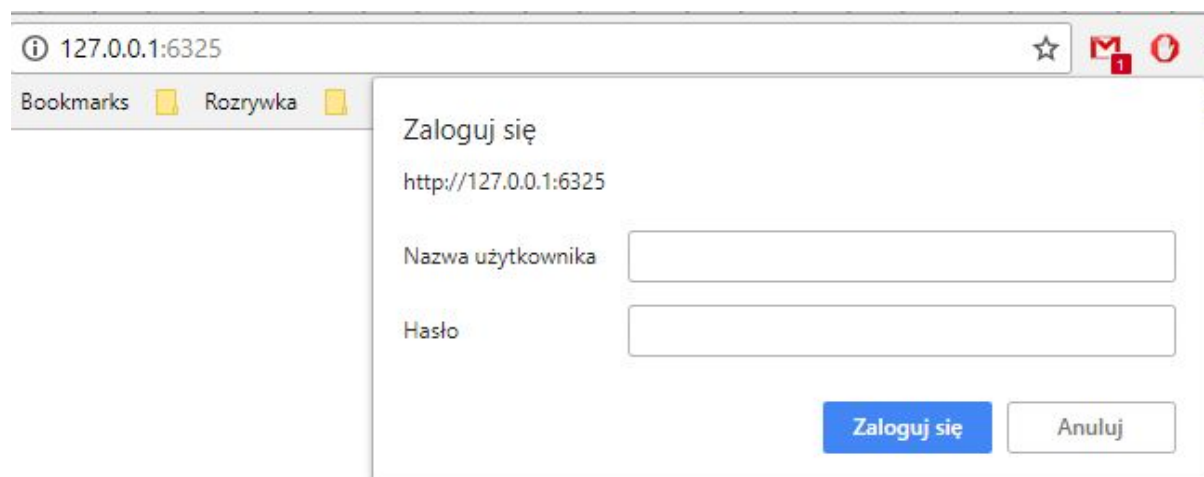
Wspólnie zajęliśmy się testami i korekcją błędów, ponieważ testowanie “nie swojego” modułu powodowało o wiele efektywniejsze wyszukiwanie błędów, niż gdy testowano “swoją” moduł. Wspólnie tworzymy również sprawozdanie z wykonania projektu.

4. Funkcjonalność aplikacji

Aplikacja jest przygotowana do pracy w systemie operacyjnym Raspbian na platformie Raspberry Pi w wersji trzeciej, bazującym na debianie. Jednak może również pracować na innych dystrybucjach linuxa, a także w systemie Windows.

W założeniach projektowych urządzenie pracuje w systemie zamkniętym i nikt nieuprawniony nie ma dostępu do sieci pozwalającej na administrację urządzeniem. Jednak na wypadek włamania wprowadziliśmy zabezpieczenia. Mianowicie przenieśliśmy panel administracyjny ze standardowego dla wszystkich aplikacji webowych portu 80 na port 6325. Jest to proste zabezpieczenie przez które niedoświadczony użytkownik nie będzie w stanie dostać się do panelu. Jeśli jednak użytkownik dostanie się na adres_urządzenia:6325 to ukaże mu się okno logowania.

Wymagające loginu oraz hasła administratora.



Rys. 1 Okno logowania do panelu administratora.

Użytkownik po uwierzytelnieniu ma dostęp do aktualnie wyświetlanego widoku. Sprawia to, że nie musi fizycznie widzieć tablicy by wiedzieć co jest aktualnie wyświetlane. Ponadto obok znajduje się tabela zdarzeń cron, która jest równocześnie kolejką wyświetlanych elementów. Pojedynczy rekord zawiera czas rozpoczęcia wyświetlania, zakończenia wyświetlania oraz etykietę typu. W tym panelu może dodawać nowe elementy, a także edytować bądź usuwać już istniejące rekordy.

Dodanie, bądź edycja nowego wydarzenia wymaga:

1. Wprowadzenia nazwy wydarzenia.
2. Wprowadzenia czasu rozpoczęcia wyświetlania.
3. Wprowadzenia czasu zakończenia wyświetlania.
4. Wprowadzenia priorytetu wyświetlania.
5. Wprowadzenia typu elementu:
 - a. Zdjęcie - wymaga dołączenia pliku.
 - b. Film - wymaga dołączenia pliku.
 - c. Zdalny adres URL - wymaga podania adresu.

W tym przypadku aplikacja robi kopie strony www co pozwala na jej wyświetlanie nawet w trybie offline.

- d. Film lub Stream na platformie YouTube - wymagany adres URL.
- e. Pasek wiadomości - wymaga:
 - Treści wiadomości.
 - Koloru tekstu.
 - Koloru tła.
- f. Zegarek wyświetlający aktualny czas.

Na dodatek cała baza wydarzeń jest przechowywana w lokalnym pliku JSON, jednak nic nie szkodzi na przeszkodzie by umieścić go na serwerze zdalnym. W takim przypadku wiele urządzeń może być zsynchronizowanych z jednego pliku. Oznacza to, że mogą wyświetlać identyczną zawartość w tym samym czasie.

5. Wybrane technologie

Przy wybieraniu technologii wzięliśmy pod uwagę wymagania funkcjonalne i niefunkcjonalne aplikacji. Szczególnie należało zwrócić uwagę na wymóg uruchomienia programu na urządzeniu Raspberry Pi.

Zdecydowaliśmy się na zastosowanie środowiska Node.js. Node.js został stworzony z myślą o tworzeniu aplikacji internetowych, szczególnie serwerów WWW. Z jego pomocą, w łatwy sposób można było stworzyć panel kontrolny aplikacji w formie strony internetowej.

Aby spełnić wymagania dotyczące wyświetlania zawartości użyliśmy frameworka Electron. Pozwala on tworzyć aplikacje z interfejsem graficznym, w których jako back-end używany jest właśnie Node.js, a jako front-end przeglądarka Chromium.

Użyliśmy następujących modułów do Node.js:

- express - framework, który praktycznie jest standardem przy tworzeniu aplikacji internetowych w środowisku Node.js. Wprowadza on między innymi potężne mechanizmy routingu, znacznie ułatwia dodawanie middleware oraz podstawowe mechanizmy serwerów HTTP takie jak przekierowania.
<https://www.npmjs.com/package/express>
- cron - moduł, który implementuje mechanizm “cron” znany z systemów Linux. Pozwala on na wywoływanie zdarzeń co dany okres zgodnie z podanym wzorem. Dodatkowo rozszerza format [crontab](#) o dokładność do sekund.
<https://www.npmjs.com/package/cron>
- Formidable - zadaniem tego modułu jest przetwarzanie danych z formularzy. Obsługuje również przysyłanie plików na serwer i ich zapisywanie.
<https://www.npmjs.com/package/formidable>
- website-scraper - moduł ten jest używany do zapisania zdalnej strony internetowej lokalnie w urządzeniu. Pobiera plik HTML, a następnie, zgodnie z konfiguracją, elementy zdalne do których odwołania znajdują się na stronie. Dzięki temu modułowi jesteśmy w stanie zapisać stronę internetową, razem z umieszczonymi na niej obrazkami i stylami, do późniejszego odtworzenia w przypadku braku połączenia z internetem
<https://www.npmjs.com/package/website-scraper>
- Basic-auth - moduł, który implementuje proste uwierzytelnienie za pomocą HTTP, opisane w [RFC761](#).
<https://www.npmjs.com/package/basic-auth>

- morgan - moduł pomocniczy, który służy do generowania rejestru wszystkich otrzymanych przez serwer żądań HTTP. Format generowanych wpisów można łatwo konfigurować, a ich obecność znacznie ułatwia wykrywanie i usuwanie błędów.
<https://www.npmjs.com/package/morgan>
- base64-img - moduł używany przy przetwarzaniu zrzutu ekranu z widoku tablicy informacyjnej. Pozwala on zapisać pobrany zrzut ekranu do pliku w formacie png.
<https://www.npmjs.com/package/base64-img>

6. Narzędzia, środowisko, biblioteki

Pełna lista wszystkich bibliotek, narzędzi, języków programowania użytych podczas implementacji.

- Aplikacja:
 - Moduł serwerowy:
 - JavaScript,
 - node.js,
 - express,
 - Formidable,
 - morgan,
 - basic-auth.
 - Moduł harmonogramu:
 - JavaScript,
 - cron,
 - website-scraper.
 - Moduł wyświetlania:
 - JavaScript,
 - Electron.js,
 - HTML5,
 - CSS,
 - Bootstrap,
 - base64-img.
- Kontrola wersji:
 - Git.
- Narzędzia:
 - Visual Studio Code,
 - nano,
 - npm,
 - Przeglądarka Google Chrome.

7. Architektura rozwiązania

Aplikację można podzielić na trzy odrębne moduły:

- Moduł internetowego panelu sterowania
Ta część aplikacji odpowiada za wystawienie serwera WWW. Udostępnia tworzenie, edycję i usuwanie zdarzeń z poziomu przeglądarki na zdalnym komputerze. Używa metod udostępnianych przez moduł harmonogramu zdarzeń do wprowadzania zmian w zdarzeniach zdefiniowanych przez użytkownika.
- Moduł obsługi harmonogramu zdarzeń
Ten moduł odpowiada za zapisywanie zdarzeń zdefiniowanych przez użytkownika. Udostępnia API do tworzenia, usuwania i aktualizowania zdarzeń. Zgodnie ze zdefiniowanym harmonogramem generuje zdarzenia wywołujące odpowiednie funkcje wyświetlania zawartości na module interfejsu graficznego.
- Moduł wyświetlania zawartości na ekranie
W tej części aplikacji została zaimplementowana obsługa interfejsu graficznego. Moduł udostępnia API umożliwiające wyświetlanie zawartości różnego rodzaju. Umożliwia również pobranie zrzutu aktualnie wyświetlanego ekranu.

8. Interesujące problemy i rozwiązania ich na jakie się natknęliśmy

Przy obsłudze danych przesyłanych na serwer skorzystaliśmy z wcześniej wspomnianego modułu Formidable. Moduł ten niestety nie obsługuje pól zawierających tabele. Pola tego typu były używane przy wyborze wartości dla miesięcy i dni tygodnia. Dla tych pól otrzymywana była tylko jedna z zaznaczonych wartości.

Problem został rozwiązany w jednym z otwartych pull requestów w serwisie GitHub:

<https://github.com/felixge/node-formidable/pull/340>

Aby uniknąć bezpośredniego modyfikowania biblioteki, przeanalizowaliśmy zmiany zaproponowane pod wyżej wymienionym adresem URL. Udało nam zaimplementować rozwiązanie, które podczas startu aplikacji “wstrzykuje” przeanalizowane zmiany do biblioteki. Po zastosowaniu tej poprawki aplikacja działa zgodnie z założeniami.

Kolejnym problemem napotkaliśmy podczas tworzenia widoku podglądu aktualnie wyświetlanej treści na stronie administratora. Electron.js odpowiadający za wyświetlanie posiada wbudowaną metodę “off-screen”. Tworzy ona wyświetlacz wirtualny o treści identycznej z ekranem głównym. Wedle dokumentacji jest to idealna metoda do uzyskania podglądu “na żywo”. Jednak z niewytłumaczalnego powodu podczas testów na Raspberry Pi podczas aktywacji tej funkcji obraz zniknął z wyświetlacza rzeczywistego jak i wirtualnego. Dlatego w finalnej wersji aplikacji posiadamy podgląd jedynie z wygenerowanego pliku .png. Generuje się on przy każdym odświeżeniu. Generowanie podglądu .png również nie należy do najprostszych. Ponieważ electron nie udostępnia do tego żadnego narzędzia. Obraz przechwytywany z okna electrona jest zapisywany w znaczniku html <video> następnie konwertowany do znacznika <canvas>. On natomiast jest zapisywany do wartości binarnych i przy pomocy base64 zapisywany do pliku .png.

Początkowo planowaliśmy, że platformą docelową dla systemu będzie Raspberry Pi zero W, jednak już w trakcie instalacji się wycofaliśmy. Ponieważ ta wersja Rpi nie wspiera bibliotek electrona. Udało się uruchomić wersję beta electrona, jednak wymagało to wielu edycji konfiguracji, a sama aplikacja nie była stabilna. Dlatego ostatecznie aplikacja działa na Raspberry Pi 3. Nie występują żadne problemy nawet podczas odtwarzania filmów Full HD.

9. Opis zapytań sieciowych

W tym punkcie zostanie przedstawiona dokumentacja wszystkich zapytań sieciowych.

UWAGA - wszystkie dane poza plikami są wymieniane w formacie JSON.

- Pobieranie harmonogramu jako listy do wyświetlenia.

GET /schedule

Parametry:

Brak

Odpowiedź:

Tabela składająca się z obiektów:

Nazwa	Typ	Opis
"name"	String	Nazwa wydarzenia.
"time"	String	Czas rozpoczęcia i zakończenia, w formacie cron do wyświetlenia.
"type"	String	Typ wydarzenia.

- Dodanie wydarzenia do harmonogramu.

POST /addSchedule

Parametry:

Nazwa	Typ	Opis
"name"	String	Nazwa wydarzenia.
"contentType"	String	Typ wydarzenia. Przyjmuje jedną z wartości: "remote" - zdalna strona "youtube" - zawartość z serwisu YouTube "video" - lokalny plik wideo "image" - lokalny plik obrazu "bar" - pasek wiadomości "clock" - zegar
"url"	String	Zdalny adres zawartości. Przyjmuje wartość dla typów "remote" oraz "youtube"
"message"	String	Zawartość wiadomości. Przyjmuje wartość

		dla typu "message"
"file"	Plik	Plik lokalny do przesłania. Przyjmuje wartość dla typów "video" oraz "image"
"startSeconds"	String	Liczba sekund do formatu cron dla startu wydarzenia.
"startMinutes"	String	Liczba minut do formatu cron dla startu wydarzenia.
"startHours"	String	Liczba godzin do formatu cron dla startu wydarzenia.
"startDoM"	String	Dni miesiąca do formatu cron dla startu wydarzenia.
"startMonths"	String	Miesiące do formatu cron dla startu wydarzenia.
"startDoW"	String	Dni tygodnia do formatu cron dla startu wydarzenia.
"stopSeconds"	String	Liczba sekund do formatu cron dla końca wydarzenia.
"stopMinutes"	String	Liczba minut do formatu cron dla końca wydarzenia.
"stopHours"	String	Liczba godzin do formatu cron dla końca wydarzenia.
"stopDoM"	String	Dni miesiąca do formatu cron dla końca wydarzenia.
"stopMonths"	String	Miesiące do formatu cron dla końca wydarzenia.
"stopDoW"	String	Dni tygodnia do formatu cron dla końca wydarzenia.
"priority"	String	Priorytet wydarzenia.
"bcolor"	String	Kolor tła tekstu. Przyjmuje wartość dla typu "message"
"color"	String	Kolor tekstu. Przyjmuje wartość dla typu "message"

- Edycja wydarzenia z harmonogramu.
[POST /editSchedule](#)

Parametry:

Nazwa	Typ	Opis
"previous"	String	Nazwa wydarzenia przed edycją.
"name"	String	Nazwa wydarzenia.
"contentType"	String	Typ wydarzenia. Przyjmuje jedną z wartości: "remote" - zdalna strona "youtube" - zawartość z serwisu YouTube "video" - lokalny plik wideo "image" - lokalny plik obrazu "bar" - pasek wiadomości "clock" - zegar
"url"	String	Zdalny adres zawartości. Przyjmuje wartość dla typów "remote" oraz "youtube"
"message"	String	Zawartość wiadomości. Przyjmuje wartość dla typu "message"
"file"	Plik	Plik lokalny do przesłania. Przyjmuje wartość dla typów "video" oraz "image"
"startSeconds"	String	Liczba sekund do formatu cron dla startu wydarzenia.
"startMinutes"	String	Liczba minut do formatu cron dla startu wydarzenia.
"startHours"	String	Liczba godzin do formatu cron dla startu wydarzenia.
"startDoM"	String	Dni miesiąca do formatu cron dla startu wydarzenia.
"startMonths"	String	Miesiące do formatu cron dla startu wydarzenia.
"startDoW"	String	Dni tygodnia do formatu cron dla startu wydarzenia.
"stopSeconds"	String	Liczba sekund do formatu cron dla końca wydarzenia.

"stopMinutes"	String	Liczba minut do formatu cron dla końca wydarzenia.
"stopHours"	String	Liczba godzin do formatu cron dla końca wydarzenia.
"stopDoM"	String	Dni miesiąca do formatu cron dla końca wydarzenia.
"stopMonths"	String	Miesiące do formatu cron dla końca wydarzenia.
"stopDoW"	String	Dni tygodnia do formatu cron dla końca wydarzenia.
"priority"	String	Priorytet wydarzenia.
"bcolor"	String	Kolor tła tekstu. Przyjmuje wartość dla typu "message"
"color"	String	Kolor tekstu. Przyjmuje wartość dla typu "message"

- Pobieranie szczegółów dotyczących danego wydarzenia.

POST /getEvent

Parametry:

Nazwa	Typ	Opis
"name"	String	Nazwa wydarzenia.

Odpowiedź:

Nazwa	Typ	Opis
"name"	String	Nazwa wydarzenia.
"type"	String	Typ wydarzenia. Przyjmuje jedną z wartości: "remote" - zdalna strona "youtube" - zawartość z serwisu YouTube "video" - lokalny plik wideo "image" - lokalny plik obrazu "bar" - pasek wiadomości "clock" - zegar
"uri"	String	Adres zawartości. Przyjmuje wartość adresu

		zdalnego dla typów "remote" oraz "youtube". I wartość adresu pliku lokalnego dla typów "video" oraz "image".
"message"	String	Zawartość wiadomości. Przyjmuje wartość dla typu "message"
"start"	String	Czas rozpoczęcia w formacie cron.
"stop"	String	Czas zakończenia w formacie cron.
"priority"	String	Priorytet wydarzenia.
"bcolor"	String	Kolor tła tekstu. Przyjmuje wartość dla typu "message"
"color"	String	Kolor tekstu. Przyjmuje wartość dla typu "message"

- Usuwanie wydarzenia z harmonogramu.

GET /deleteEvent

Parametry:

Nazwa	Typ	Opis
"name"	String	Nazwa wydarzenia do usunięcia.

Odpowiedź:

Brak

- Pobieranie aktualnego podglądu wyświetlanego obrazu.

GET /currentScreen

Parametry:

Brak

Odpowiedź:

Plik zrzutu ekranu. Obraz w formacie png.

10. Instrukcja użytkowania aplikacji

Pierwszym krokiem jest instalacja systemu Raspbian na Raspberry Pi 3. Następnie po konfiguracji interfejsów sieciowych oraz kont użytkowników przechodzimy do instalacji komponentów niezbędnych dla naszej aplikacji oraz niej samej. Korzystamy z ulubionego menadżera paczek, w prezentowanym przykładzie używamy *apt*.

Wprowadzamy następujące komendy do terminala:

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

```
sudo apt-get install nodejs
```

```
sudo apt-get install npm
```

```
git clone https://github.com/hobitolog/InfoBoard.git
```

```
cd InfoBoard
```

```
npm install
```

```
npm install -g electron
```

```
sudo ufw allow 6325/tcp
```

Za pomocą komendy *electron main.js* możemy uruchomić aplikację.

Następnie dodajemy aplikację do autostartu, można to zrobić w sposób opisany poniżej.

W lokalizacji `/etc/systemd/system/` tworzymy plik dla nowego serwisu.

Nazwijmy go `infoboard.service`

Plik powinien mieć następującą zawartość:

```
[Unit]
Description=InfoBoard Service
After=network.target

[Service]
Environment="NODE_ENV=production"
Type=simple
User=ubuntu
ExecStart=/usr/bin/electron <lokalizacja>

[Install]
WantedBy=multi-user.target
```

Gdzie zamiast `<lokalizacja>` podajemy ścieżkę do katalogu, gdzie znajdują się pliki projektu. Po wprowadzeniu zmian do plików serwisów należy powiadomić system o nowych zmianach za pomocą polecenia:

```
sudo systemctl daemon-reload
```

Od teraz możemy uruchamiać i zatrzymywać naszą aplikację za pomocą poleceń:

```
sudo systemctl start infoboard.service
```

```
sudo systemctl stop infoboard.service
```

Jeśli chcemy włączyć/wyłączyć uruchamianie aplikacji przy starcie systemu musimy użyć poleceń:

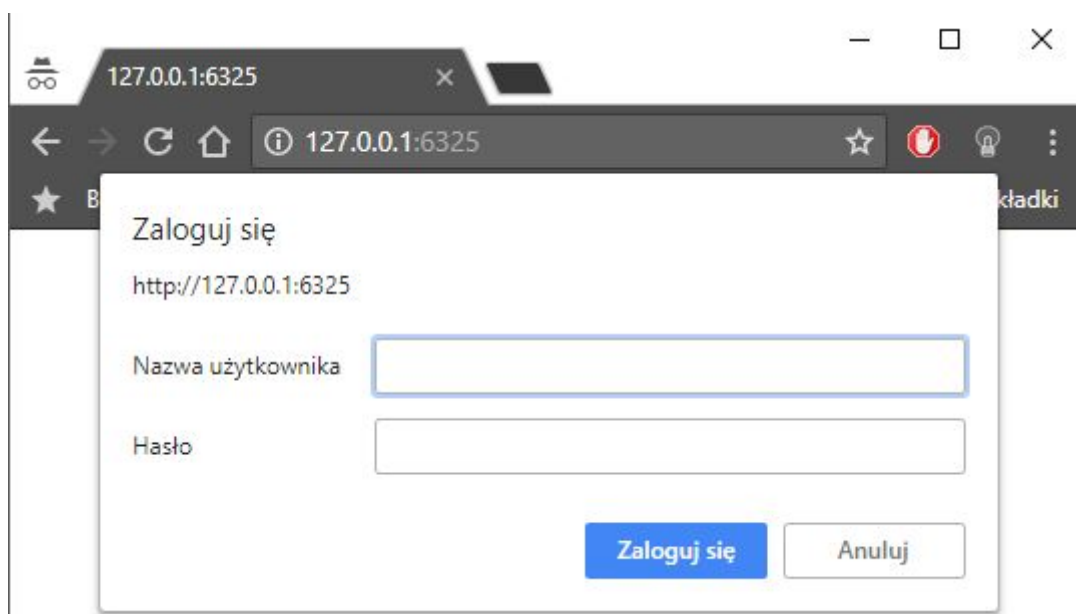
```
sudo systemctl enable infoboard.service
```

```
sudo systemctl disable infoboard.service
```

W każdej chwili możemy sprawdzić status naszego serwisu za pomocą polecenia:

```
sudo systemctl status infoboard.service
```

Teraz gdy nasza aplikacja startuje wraz z systemem możemy przejść do panelu administratora w celu konfiguracji harmonogramu. Robimy to przez przejście pod adres IP urządzenia, dodanie :6325. Dla przykładu *127.0.0.1:6325*.



Rys. 2 Widok przed zalogowaniem.

Logujemy się za pomocą danych zakodowanych w pliku *server.js*.

Po zalogowaniu uwierzytelniony użytkownik widzi pustą kolejkę wydarzeń oraz podgląd aktualnego wyświetlania. Domyślnym tłem jest zaproszenie do gry przeglądarkowej stworzonej przez twórców aplikacji.



Rys. 3 Widok po zalogowaniu.

Z tego miejsca przechodzimy do dodawania wydarzenia. Robimy to przez naciśnięcie przycisku “*Add new event*”. Przekierowuje nas to do nowej strony, gdzie podajemy nazwę wydarzenia, a także czasy rozpoczęcia i zakończenia zgodnie z notacją [cron](#). Dodatkowo wprowadzamy priorytet wyświetlania z przedziału 1-100.

Add new event to schedule:

Name

EXAMPLE_NAME

Start Time

Hours

Minutes

Seconds

Day of Month

Months

Days of Week

12

0-59

0-59

1-31

January

February

March

April

Monday

Tuesday

Wednesday

Thursday

Stop Time

Hours

Minutes

Seconds

Day of Month

Months

Days of Week

12

30

0-59

1-31

January

February

March

April

Monday

Tuesday

Wednesday

Thursday

Content Type

Upload image

Upload video

Remote URL

Youtube URL

Message Bar

Clock

Wybierz plik

Nie wybrano pliku

Priority

67

Cancel

Add to Schedule

Rys. 4 Wprowadzenie podstawowych danych.

Następnie przechodzimy do najważniejszego, czyli wyboru typu materiału jaki planujemy zamieścić. W zależności od typu wprowadzamy inne dane. Mogą to być pliki, linki do stron i filmów lub tekst.

Content Type

Upload image

Upload video

Remote URL

Youtube URL

Message Bar

Clock

Wybierz plik

Nie wybrano pliku

Rys. 5 Dodawanie zdjęcia.

Content Type

Upload image Upload video Remote URL Youtube URL Message Bar Clock

Wybierz plik Nie wybrano pliku

Rys. 6 Dodawanie filmu.

Content Type

Upload image Upload video Remote URL Youtube URL Message Bar Clock

Enter URL

Rys. 7 Dodawanie adresu zdalnego.

Content Type

Upload image Upload video Remote URL Youtube URL Message Bar Clock

Enter URL

Rys. 8 Dodawanie adresu do filmu lub strumienia Youtube.

W przypadku dodawania paska wiadomości poza samą wiadomością musimy sprecyzować kolor wyświetlanego tekstu oraz tła. Dostępna jest paleta RGB oraz HSV. Należy pamiętać iż kolor może wyglądać inaczej na różnych wyświetlaczach.

Content Type

Upload image Upload video Remote URL Youtube URL Message Bar Clock

Enter message

Background Color: Text Color:

Rys. 9 Dodawanie paska wiadomości.

Podczas dodawania zegarka nie musimy podawać, żadnej dodatkowej konfiguracji. Będzie on wyświetlony w lewym górnym narożniku w kolorze czarnym.

Content Type

Upload image Upload video Remote URL Youtube URL Message Bar Clock

Rys. 10 Dodawanie zegarka.

Następnym krokiem jest zatwierdzenie wprowadzonego eventu za pomocą przycisku “Add to Schedule”. Jeśli wypełniliśmy wszystkie pola poprawnie to zostaniemy przeniesieni z powrotem do ekranu głównego. W innym wypadku aplikacja zakomunikuje nam, co wypełniliśmy źle.

Zakładając sukces dodania wydarzenia. Ukaże nam się tabela gdzie każdy rekord pokazuje czas rozpoczęcia oraz zakończenia, a także etykietę typu.

Scheduled events:		
EXAMPLE_NAME	* 45 23 * * * * 30 12 * * *	Message Bar
obraz	12 * 12 2 2 * 2 2 * 2 * 2	Image
video	* * 1 12 * 3 * * 3 * * *	Video
kosert	* * * 24 * * * * * 30 * *	Remote URL
video_yt	* 22 * * * * * 25 * * * *	Youtube
video_yt	23 23 * * * * 12 26 * * * *	Youtube
time	* * * * * * * * * * * *	Clock
Add new event		

Rys. 11 Kolejka wydarzeń.

Jak widać na powyższym rysunku mamy dwa wydarzenia związane z Youtube. Jest to sytuacja jak najbardziej dozwolona, jednak w celu dydaktycznym założymy, że chcemy jeden z nich usunąć. W tym celu klikamy na dany rekord, a następnie na czerwony przycisk “Delete” u dołu.

Edit event:

Name

video_yt

Start Time

Hours	Minutes	Seconds	Day of Month	Months	Days of Week
0-23	22	0-59	1-31	January February March April	Monday Tuesday Wednesday Thursday

Stop Time

Hours	Minutes	Seconds	Day of Month	Months	Days of Week
0-23	25	0-59	1-31	January February March April	Monday Tuesday Wednesday Thursday

Content Type

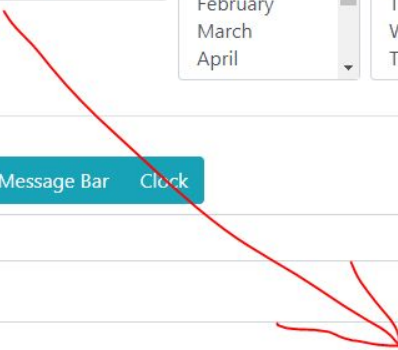
Upload image Upload video Remote URL Youtube URL Message Bar Clock

<https://www.youtube.com/watch?v=QH2-TGULwu4>

Priority

34

Cancel Save Delete

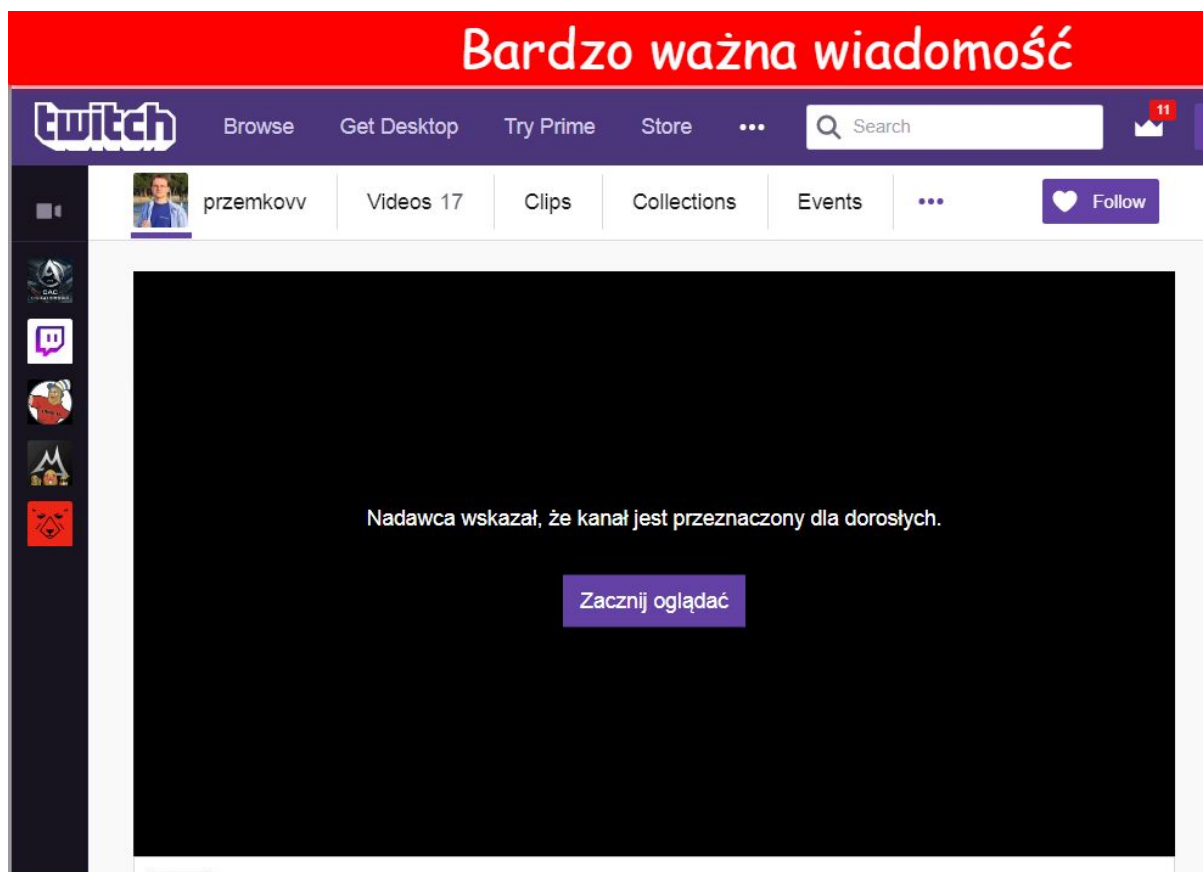


Rys. 12 Usuwanie wydarzenia.

Jak widać na rysunku 12 po kliknięciu na rekord w tabeli wydarzeń przechodzimy do edycji danego wydarzenia.

11. Wyświetlane widoki

Na ekranie aplikacji wyświetlana jest zawartość zgodna ze zdarzeniami zdefiniowanymi w harmonogramie. Jeżeli pasek wiadomości jest aktywny to zostanie on wyświetlony na górze ekranu. Jego kolory i tekst będą zgodne ze zdefiniowanymi w zdarzeniu, a treść będzie przewijać z się z prawej strony na lewą.



Rys. 14 Widok ekranu przy aktywnych zdarzeniach paska wiadomości i wyświetlania zdalnej strony internetowej <https://www.twitch.tv/przemkovv>.

Jeżeli w danym momencie nie jest aktywne żadne zdarzenie wyświetlania zawartości to wyświetlany jest obraz zastępczy. Domyślnie tym obrazem jest reklama innego tworzonego przez nas projektu. Przy aktywnym zdarzeniu zegara pokazuje się on z lewej strony pod paskiem wiadomości.



Rys. 14 Widok ekranu z aktywnym zegarem i paskiem wiadomości oraz brakiem aktywnych zdarzeń zawartości.

12. Możliwości rozwoju

Projekt posiada wiele możliwości rozwoju. Najważniejszą jest ścieżka synchronizacji wyświetlania na większej ilości urządzeń oraz mechanizm kontroli aktualności harmonogramu wyświetlania. W tym wypadku można również rozważyć kwestie zwiększenia bezpieczeństwa przykładowo poprzez dodanie szyfrowania danych, przesyłanych między urządzeniami.

Projekt możemy rozwinąć o więcej wyświetlanych typów. Poza transmisjami z YouTube oraz filmami z tego serwisu, możemy wprowadzić transmisje z innych serwisów streamingowych na przykład twitch.com oraz odtwarzania filmów z platform vimeo, facebook i innych.

Kolejnym możliwym aspektem jest wprowadzenie dostępu dla użytkowników z różnymi uprawnieniami. Przykładowo właściciela wydarzenia, który może edytować jedynie swój event, założymy treść paska informacyjnego w przypadku dezaktualizacji treści. Odbywałoby się to szybciej, niż przy każdorazowej zmianie przez administratora platformy. Czas wymagany na zgłoszenie, rozpatrzenie oraz zmianę mógłby powodować dezaktualizację aktualizowanej informacji.

W module podglądu aktualnie wyświetlanej treści obecnie wyświetlamy zrzut ekranu wytwarzany przy odświeżeniu podglądu. Jest to rozwiązanie wystarczające, jednak moglibyśmy wprowadzić podgląd na żywo. Znacznie poprawiłoby to wygląd interfejsu administratora, jednak byłoby to obciążenie już nie wyrabiającej się przy wyższych rozdzielczościach filmów platformy Raspberry Pi. Jeszcze ciekawszą koncepcją jest rozwinięcie podglądu za pomocą pulpitu zdalnego. Jednak ta opcja wymaga znacznych zasobów oraz kolejnych zabezpieczeń.

Interfejs użytkownika może w przyszłości zostać rozbudowany o aplikację mobilną wspieraną przez wiodące mobilne systemy operacyjne, czyli iOS oraz Android.

13. Podsumowanie

InfoBoard jest już trzecim projektem, w którym zespół wystąpił we wcześniej zaprezentowanym składzie. Wcześniejsza współpraca pozwoliła na wzajemne poznanie umiejętności członków zespołu. Dzięki zróżnicowaniu zainteresowań drużyna posiada szeroki wachlarz umiejętności.

We wczesnej fazie projektowania, tuż po wyborze tematu zespół rozpoczął dyskusję na temat wymagań stawianych aplikacji. Wymagania wskazują kierunek, w który powinien zwrócić się zespół podczas wyboru technologii. Po dokładnej analizie wymagań stawianych przez prowadzącego podjęto decyzję wyboru czegoś lekkiego, konkretniej mówiąc języka JavaScript w środowisku node.js.

Node.js pozwala na łatwe projektowanie aplikacji z interfejsem webowym. Dodatkowo ze względu na swoją popularność zapewnia szerokie wsparcie społeczności oraz wiele bibliotek i frameworków bogatych w różnorodne funkcjonalności. Ważnym czynnikiem przy jego wyborze była również jego wieloplatformowość.

Po wyborze technologii zespół zdecydował o architekturze rozwiązania. Aplikację z najogólniejszego punktu widzenia można podzielić na dwie części. Z pierwszej z nich, części sterującej można wydzielić moduł obsługi harmonogramu zdarzeń oraz moduł webowy, służący do zarządzania harmonogramem, do jego tworzenia. Drugą część składa się wyłącznie z jednego modułu, modułu wyświetlającego, który wyświetla “zdarzenia” zawarte w harmonogramie. Informacje na temat zawartości do wyświetlenia dostaje z części sterującej aplikacji.

Następnym krokiem było podzielenie wcześniej wspomnianych modułów na pomniejsze zadania, które można przydzielić do konkretnych osób. Przydział zadań nie był losowy, decydowały o nim umiejętności członków zespołu oraz chęć nabycia przez nich nowych umiejętności. Po przydziale zadań ustalono luźny harmonogram prac, którego głównym zadaniem było wskazanie kolejności implementacji kolejnych fragmentów aplikacji.

Faza implementacyjna przebiegła bez większych problemów. Po implementacji kolejnych funkcjonalności członkowie zespołu wzajemnie testowali swoje rozwiązania. W przypadku wystąpienia nieoczekiwanych lub błędnych zachowań rozwiązania było odrzucane i osoba odpowiedzialna za implementację danego fragmentu kodu musiała wprowadzać poprawki. Rozwiązanie było akceptowane wyłącznie wtedy, gdy działało zgodnie z wcześniejszymi założeniami.

Dzięki regularnej pracy zespołu projekt został ukończony przed ostatecznym terminem. Prezentacja funkcjonalności aplikacji odbyła się zgodnie z założeniami na platformie Raspberry Pi w wersji trzeciej. Podczas prezentacji nie wystąpiły żadne problemy, zespół odpowiadał na pytania publiczności, a prowadzący uznał projekt za spełniający stawiane mu wymagania.

14. Źródła wiedzy

- <https://nodejs.org/dist/latest-v8.x/docs/api/>
- <https://expressjs.com/en/api.html>
- <https://electronjs.org/docs>
- <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>
- <http://crontab.org/>
- <https://getbootstrap.com/docs/4.0/getting-started/introduction/>
- <https://stackoverflow.com/>