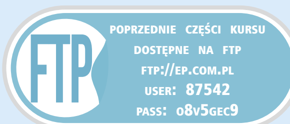
Programowanie aplikacji mobilnych (7)

Multimedia



Nawet przeciętny smartfon to urządzenie typowo multimedialne. Łączy w sobie wiele elementów, pozwalających na nagrywanie i odtwarzanie dźwięków oraz obrazów. Korzystając z jednego smartfonu można zastąpić niemało urządzeń wejścia i wyjścia, a dodatkowo używając jego potencjał komunikacyjny, zbierane dane można przesyłać i zdalnie je przetwarzać lub nimi zarządzać. W niniejszej części kursu pokazujemy, jak to zrobić, na przykładzie prostego, scentralizowanego systemu bezpieczeństwa, w którym smartfon jest urządzeniem wejściowym.

Nasz nowy przykład będzie podobny do podawanego wcześniej przykładu sterowania bramą, z tą różnicą, że skoncentrujemy się na innych elementach niż dotychczas. Za pomocą aparatu wbudowanego w telefon będziemy rejestrowali obraz, a z użyciem mikrofonu – także i dźwięk, które poprzez Internet będziemy przesyłać do głównego serwera, celem ich weryfikacji. Gdy weryfikacja będzie poprawna, smartfon powita użytkownika. Sama weryfikacja będzie leżała po stronie centralnego systemu, a więc nie będziemy implementowali jej na telefonie i nie opiszemy jej w tym przykładzie. Jej realizację można oprzeć np. o biblioteki OpenCV i stosowny system bazodanowy, ale wykracza to poza zakres niniejszego kursu.

Dodatkowo, aby pokazać jak zastosować dwie przydatne wtyczki, procedurę sprawdzania tożsamości gościa będziemy rozpoczynali od zeskanowania jego karty z identyfikatorem QR, (co uprości zadanie serwera), a całość komunikatów będziemy podawać głosowo za pomocą syntezatora mowy.

Nowe wtyczki i nowa wersja środowiska

Aby niniejszy kurs był jak najdłużej aktualny, jesteśmy zmuszeni zaktualizować nasze środowisko do nowszej wersji, która korzysta z nowego systemu wtyczek. Wspominaliśmy już o nich w 5. części tego kursu, opublikowanej w lipcowym numerze Elektroniki Praktycznej.

Zaczynamy od aktualizacji Cordovy do najnowszej wersji. Należy się spodziewać, że osoby, które dopiero teraz rozpoczynają śledzenie kursu, jeśli postarają się pobierać aktualne wersje oprogramowania Cordovy, będą je miały w tej samej wersji, w której my będziemy mieli za chwilę. Wykonujemy polecenie *npm update –g cordova*, które uaktualni środowisko Cordova do najnowszej wersji (w naszym przypadku 5.3.1). W trakcie aktualizacji pojawi się zapewne komunikat o przestarzałych wersjach samego serwera node.js i menedżera pakietów npm. My na początku kursu zainstalowaliśmy node.js w wersji 0.12.0, który zawierał program npm w wersji 1.4.28. Od tego czasu twórcy node.js znacznie przyspieszyli z numeracją i aktualnie najnowsza wersja tego oprogramowania to 4.1.0. Pobieramy ją ze strony nodejs.org i instalujemy, jak każdy program, z domyślnymi ustawieniami.

Po aktualizacji możemy sprawdzić, które wersje mamy zainstalowane, korzystając z poleceń:

```
node -v
npm -v
cordova -v
```

Jeśli wszystko poszło dobrze, powinniśmy mieć na komputerze zainstalowane przynajmniej wersje 4.1.0, 2.14.3 i 5.3.1 (odpowiednio), lub nowsze. Ponieważ tę część kursu zaczynamy od czystego projektu, nie musimy aktualizować platformy androidowej.

Tworzymy nowy projekt i dodajemy do niego platformę Android – w naszym przypadku zainstalowała się wersja 4.1.1. Warto zauważyć, że wraz z tą wersją platformy automatycznie instaluje się plugin **cordova-plugin-whitelist** (w naszym przypadku w wersji 1.0.0). Jest to konieczna nowość, zmieniająca sposób zabezpieczenia aplikacji przed odwoływaniem się pod niepożądane adresy internetowe.

Z uwagi na potrzeby tej części kursu instalujemy od razu użyteczne dla nas wtyczki:

- **cordova-plugin-file-transfer**
- **cordova-plugin-media**
- **cordova-plugin-camera**
- **org.apache.cordova.speech.speechsynthesis**
- **phonegap-plugin-barcodescanner**

W trakcie instalacji pierwszej z wtyczek, system automatycznie pobierze wymaganą wtyczkę **cordova-plugin-file**. Wersje zainstalowanych pluginów, jakie uzyskaliśmy w naszym przypadku, podajemy w **tabeli 1**. Można je sprawdzić u siebie, korzystając z polecenia *cordova plugin*. Po aktualizacji oprogramowania i dodaniu nowej, aktualnej platformy Android do projektu, domyślną wersją API projektu jest 22, czyli odpowiadająca Androidowi 5.1.1 (Lollipop). Może się zdarzyć (np. jeśli ktoś dokładnie podążał kursem, trzymając się tych samych wersji oprogramowania), że nie będzie miał zainstalowanej tej wersji API. Gdy rozpoczynaliśmy kurs najnowszym dostępnym API było 21, a obecnie jest to już 23 (Android 6.0 Marshmallow). Aby pobrać nowe SDK konieczne jest uruchomienie programu Android SDK Manager, który można wywołać z Android Studio, poprzez naciśnięcie *File → Settings → Appearance & Behavior*

Tabela 1. Plugin pobrane na potrzeby niniejszej części kursu i ich wersje. Warto zwrócić uwagę, że plugin SpeechSynthesis nie został jeszcze przeniesiony do nowego repozytorium npm i cordova nie znajdując go tam, poszukała go w starym repozytorium

Nazwa	Plugin	Wersja
Camera	cordova-plugin-camera	1.2.0
File	cordova-plugin-file	3.0.0
File Transfer	cordova-plugin-file-transfer	1.2.1
Media	cordova-plugin-media	1.0.1
Whitelist	cordova-plugin-whitelist	1.0.0
SpeechSynthesis	org.apache.cordova.speech.speechsynthesis	0.1.0
BarcodeScanner	phonegap-plugin-barcodescanner	4.0.2

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 87542, pass: o8v5gec9

→ *System Settings* → *Android SDK* lub klikając *Tools* → *Android* → *SDK Manager*, albo też z linii poleceń, wpisując samo polecenie *android*.

Po załadowaniu się informacji o dostępnych aktualizacjach zaznaczamy wybrane API. W naszym przypadku pobieramy tylko API 22 oraz pozwalamy na aktualizację wszystkich pozostałych narzędzi, których nowsze wersje się pojawiły. Warto to zrobić, gdyż często zawierają one usprawnienia oraz są pozbawione błędów znalezionych w międzyczasie. Proces aktualizacji może potrwać nawet kilkadziesiąt minut, a pierwsze kompilowanie nowych projektów po wszystkich tych aktualizacjach również potrwa dłużej, gdyż zaktualizowane zostaną dodatkowe potrzebne narzędzia i biblioteki.

Procedura działania projektowanego urządzenia

Na potrzeby przykładu, opracowaliśmy następujący mechanizm działania – użytkownik podchodzi do stale włączonego smartfonu z Androidem. Naciska przycisk „otwórz", po czym uruchamia się kamera w trybie skanowania kodów QR. Użytkownik zbliża swój identyfikator z kodem i gdy kod zostanie poprawnie rozpoznany, kamera przełącza się w tryb robienia zdjęcia. Użytkownik ustawia się tak, by jego twarz była widoczna przez obiektyw i wykonuje zdjęcie. Następnie użytkownik przedstawia się, a jego próbka głosu jest rejestrowana. Zebrane dane, tj. zdjęcie twarzy i próbka głosu przesyłane są na zdalny serwer, do katalogu zależnego od numeru odczytanego z kodu QR na identyfikatorze użytkownika. Serwer stara się jak najszybciej sprawdzić, czy zdjęcie i głos pasują do danego, zapisanego w bazie użytkownika i generuje odpowiedź – tej części programu nie realizujemy w ramach kursu i zakładamy,, że działa ona poprawnie (stworzyliśmy serwer, który automatycznie zawsze odpowiada poprawnie). Jeśli odpowiedź jest pozytywna, smartfon łączy się z innym serwerem i pobiera spersonalizowany ekran powitalny (lub np. ustawienia) danego użytkownika i go wyświetla. Poszczególnym etapom rozpoznawania użytkownika towarzyszą wskazówki podawane głosowo przez smartfon.

Rozpoznawanie kodów QR

Zaczynamy od rozpoznawania kodów QR, zapisanych na identyfikatorach użytkowników. W tym celu korzystamy z wygodnej wtyczki **phonegap-plugin-barcodescanner**. To bardzo proste narzędzie – ma tylko dwie funkcje:

- **cordova.plugins.barcodeScanner.scan(success, fail)** – umożliwiającą zeskanowanie kodu i wywołanie funkcji **success** w przypadku powodzenia, lub funkcji **fail**, gdy się nie uda,
- **cordova.plugins.barcodeScanner.encode(type, data, success, fail)** – umożliwiającą wygenerowanie kodu o zadanym typie (**type**), na podstawie podanych danych (**data**) i następnie uruchomienie funkcji **success** lub **fail**, w zależności od powodzenia zadziałania.

Funkcja skanująca uruchamia kamerę smartfona z zaznaczonym obszarem, w którym powinien znaleźć się rozpoznawany kod QR. Zostało to przedstawione na **rysunku 1**. Co ważne, plugin ten pozwala wczytywać różne rodzaje kodów, a nie tylko QR. Ich lista została zebrana w **tabeli 2**

Tabela 2. Lista kodów graficznych rozpoznawana przez plugin phonegap-plugin-barcodescanner, w zależności od systemu operacyjnego

Kod\Platforma	Android	iOS	Windows 8	Windows Phone 8	BlackBerry 10
AZTEC	NIE	NIE	TAK	TAK	TAK
CODABAR	TAK	NIE	TAK	TAK	NIE
CODE_128	TAK	TAK	TAK	TAK	TAK
CODE_39	TAK	TAK	TAK	TAK	TAK
CODE_93	TAK	NIE	TAK	TAK	NIE
DATA_MATRIX	TAK	TAK	TAK	TAK	TAK
EAN_13	TAK	TAK	TAK	TAK	TAK
EAN_8	TAK	TAK	TAK	TAK	TAK
ITF	TAK	TAK	TAK	TAK	TAK
MSI	NIE	NIE	TAK	TAK	NIE
PDF417	TAK	NIE	TAK	TAK	NIE
QR_CODE	TAK	TAK	TAK	TAK	NIE
RSS_EXPANDED	TAK	NIE	NIE	NIE	NIE
RSS14	TAK	NIE	TAK	TAK	NIE
UPC_A	TAK	TAK	TAK	TAK	TAK
UPC_E	TAK	TAK	TAK	TAK	TAK

i jest zależna od systemu operacyjnego, na którym uruchomiona jest aplikacja.

Funkcja tworząca kod QR (i tylko QR) przyjmuje jako typ jeden z czterech rodzajów:

- **TEXT_TYPE**,
- **EMAIL_TYPE**,
- **PHONE_TYPE**,
- **SMS_TYPE**.

W zależności od typu wygenerowanego kodu, będzie on inaczej obsługiwany przez różne aplikacje, które go wczytają.

My będziemy tylko skanowali kod i jeśli okaże się on mieć poprawny format, przejdziemy do fotografowania twarzy. Poprawność kodu sprawdzamy poprzez porównanie wartości atrybutów *text* i *format*, zmiennej zwracanej jako parametr wywołania funkcji w przypadku powodzenia skanowania kodu.

Korzystanie z kamery

Zdjęcie użytkownika wykonamy korzystając z pluginu **cordova-plugin-camera**. Pozwala on nie tylko wykonać

Rysunek 1. Skanowanie kodu z użyciem wtyczki *phonegap-plugin-barcodescanner*. Pozwala ona nie tylko wczytywać kody QR, ale też różne formaty kodów kreskowych, tak jak np. kod EAN_13, umieszczany na okładce Elektroniki Praktycznej

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 87542, pass: o8v5gec9

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 87542, pass: o8v5gec9

nowe fotografie, ale też dać użytkownikowi wybór zdjęcia z galerii oraz obejmuje kilka dodatkowych opcji.

Po instalacji pluginu, w JavaScripcie, w obiekcie **navigator** pojawia się nowy atrybut **camera**, również będący obiektem i zawierający dwie funkcje (w tym jedną tylko dla systemu iOS) oraz dwa obiekty z samymi stałymi i jedną zmienną. Oto one:

- **getPicture(success, fail, options)** – funkcja wykonująca zdjęcie, lub pozwalająca użytkownikowi na wybranie go z galerii (zależnie od parametru **options**). Po pomyślnym zrobieniu lub wybraniu fotografii uruchamiana jest funkcja *success*, której parametrem jest nowy obraz w odpowiednim formacie, ustawionym w opcjach. W przeciwnym wypadku wykonywana jest funkcja *fail*.
- **cleanup(success, fail)** – funkcja czyszcząca zdjęcia w pamięci tymczasowej, działająca tylko w systemie iOS.
- **CameraOptions** – obiekt zawierający opcje (stałe), jakie można przekazać do funkcji **getPicture()**, by określić zasady wykonywania lub wybierania zdjęć.
- **CameraPopoverHandle** – zmienna, do której przypisywana jest funkcja wyświetlająca okno dialogowe w momencie uruchomienia funkcji **getPicture()**; zmienna ta jest użyteczna tylko w systemie iOS.
- **CameraPopoverOptions** – obiekt zawierający opcje (stałe), jakie można przekazać w ramach dodatkowych opcji wewnątrz **CameraOptions** funkcji **getPicture()**, na potrzeby uruchomienia funkcji wskazanej przez **CameraPopoverHandle**; element ten jest użyteczny tylko w przypadku systemu iOS.

Ponieważ aplikację testujemy na Androidzie, skoncentrujemy się tylko na funkcji *navigator.camera.getPicture()* i opcjach dostępnych w ramach *navigator.camera.CameraOptions*.

Obiekt *navigator.camera.CameraOptions* zawiera następujące atrybuty:

- **quality** – wartość od 0 do 100, określająca pożądaną jakość wykonywanych zdjęć. Domyślnie wartość tego parametru wynosi 50, przy czym plugin nie pozwala z góry sprawdzić parametrów wbudowanej kamery.
- **destinationType** – to bardzo ważny parametr, bo określa, w jakim formacie chcemy otrzymać wykonane zdjęcie. Może to być:
  - **DATA_URL**, czyli string zakodowany algorytmem base64, zawierający dane binarnej fotografii,
  - **FILE_URI**, czyli adres URI pliku z wykonanym zdjęciem; jest to domyślny typ,
  - **NATIVE_URI**, czyli adres pliku podany zgodnie ze schematem katalogów z danymi, używanym w danym systemie operacyjnym;
- **sourceType** – parametr ten pozwala określić, czy zdjęcie ma być wybrane z:
  - biblioteki (wartość **PHOTOLIBRARY**),
  - aparatu (wartość **CAMERA** – domyślna),
  - wbudowanej pamięci (wartość **SAVEDPHOTOALBUM**);
- **allowEdit** – informuje, czy użytkownik może dokonać prostej obróbki zdjęcia przed jego przekazaniem do aplikacji (domyślnie nie);
- **encodingType** – określa rodzaj kompresji (domyślnie **JPEG**, alternatywnie **PNG**);
- **targetWidth** – określa szerokość, do której obraz ma być przeskalowany (domyślnie bez skalowania); parametru tego należy użyć tylko w połączeniu z użyciem parametru **targetHeight**;
- **targetHeight** – określa wysokość, do której obraz ma być przeskalowany (domyślnie bez skalowania); parametru tego należy użyć tylko w połączeniu z użyciem parametru **targetWidth**;
- **mediaType** – określa nośnik, z którego ma być wybrane zdjęcie, jeśli parametr **sourceType** wskazuje na inną opcję niż **CAMERA**; dostępne są trzy opcje: **PICTURE**, **VIDEO** (nagranie wideo) i **ALLMEDIA**;
- **correctOrientation** – parametr określający, czy obraz ma być obrócony tak, by jego orientacja była zgodna z orientacją aparatu, w trakcie robienia zdjęcia;
- **saveToPhotoAlbum** – parametr określający, czy zdjęcie wykonane kamerą ma być zapisane w systemowej galerii;
- **popoverOptions** – dodatkowe ustawienia, określone za pomocą stałych zdefiniowanych w obiekcie **CameraPopoverOptions**;
- **cameraDirection** – opcja umożliwiająca wybór kamery, za pomocą której ma być wykonywane zdjęcie; dostępne są opcje **BACK** i **FRONT**; domyślną jest **BACK**; niestety parametr ten jest w praktyce słabo obsługiwany.

Nagrywanie dźwięku

Do nagrania głosu użytkownika użyjemy pluginu **cordova-plugin-media**. Udostępnia on w programie klasę *Media*, której obiekty możemy tworzyć i korzystać z nich do rejestracji lub odtwarzania dźwięku. Konstruktor klasy *Media* przyjmuje cztery parametry:

- **src** – adres URI, pod którym znajduje się lub ma się znajdować nagranie;
- **mediaSuccess** – opcjonalna funkcja, która ma być wywołana po pomyślnym zakończeniu odtwarzania, nagrywania lub wstrzymania odtwarzania nagrania;
- **mediaError** – opcjonalna funkcja, która ma być wywołana w przypadku wystąpienia błędu w obsłudze nagrania;
- **mediaStatus** – opcjonalna funkcja, uruchamiana gdy stan nagrania się zmieni. Funkcji tej przekazywany jest parametr o jednej z następujących wartości:
  - **MEDIA_NONE** = 0,
  - **MEDIA_STARTING** = 1,
  - **MEDIA_RUNNING** = 2,
  - **MEDIA_PAUSED** = 3,
  - **MEDIA_STOPPED** = 4.

Po utworzeniu obiektu klasy **Media** możemy skorzystać z jego 10 metod i 2 atrybutów dostępnych tylko do odczytu:

- **getCurrentPosition()** – funkcja zwraca aktualną pozycję w pliku audio,
- **getDuration()** – funkcja zwraca łączną długość pliku audio,
- **play()** – funkcja rozpoczyna lub wznawia odtwarzanie pliku audio,
- **pause()** – funkcja pauzuje odtwarzanie pliku audio,
- **release()** – funkcja zwalnia zasoby audio systemu operacyjnego,
- **seekTo()** – funkcja przeskakuje do określonej pozycji w pliku audio,
- **setVolume()** – funkcja pozwala ustawić głośność odtwarzania,
- **startRecord()** – funkcja uruchamia rejestrację nagrania audio do pliku,
- **stopRecord()** – funkcja zatrzymuje rejestrację nagrania audio do pliku,
- **stop()** – funkcja zatrzymuje odtwarzanie pliku audio,
- **position** – atrybut wyrażający w sekundach aktualną pozycję w pliku audio,
- **duration** – atrybut wyrażający w sekundach łączną długość pliku audio.

Funkcja *media.getCurrentPosition()* przyjmuje jako parametry kolejno nazwy funkcji sukcesu i porażki. Funkcja *media.getDuration()* nie przyjmuje parametrów, a wyniki zwraca wyrażone w sekundach. Funkcja *media.pause()*, *media.play()*, *media.release()*, *media.stop()*, *media.startRecord()* i *media.stopRecord()* standardowo nie przyjmują parametrów ani nie zwracają żadnych wartości. Funkcja *media.seekTo()* przyjmuje jako parametr pozycję w pliku, wyrażoną w milisekundach. Funkcja *media.setVolume()* przyjmuje jako parametr wartość z zakresu od 0 do 1.

Aby nagrać głos użytkownika, tworzymy nowy obiekt klasy *Media*, wskazując jego ścieżkę oraz definiując funkcję sukcesu, tak by móc obsłużyć powstały plik. Następnie włączamy funkcję *media.startRecord()* i za pomocą funkcji *setTimeout()* ustawiamy wywołanie funkcji *media.stopRecord()* np. po 4 sekundach. Na koniec operacji z dźwiękiem wywołujemy funkcję *media.release()*, która jest istotna szczególnie na Androidzie i pozwala zwolnić używane zasoby.

Przesyłanie danych na serwer

Gdy mamy już wykonane zdjęcie i nagrany głos użytkownika, przesyłamy je na serwer. Tym razem wykorzystamy

w tym celu wygodną wtyczkę **cordova-plugin-file-transfer**. Ma ona jeden atrybut i trzy funkcje:
- **upload()** – funkcja nakazująca przesłanie pliku na serwer,
- **download()** – funkcja nakazująca pobranie pliku z serwera,
- **abort()** – funkcja przerywająca transfer,
- **onprogress** – atrybut, któremu przypisuje się nazwę funkcji do uruchomienia w momencie gdy nowa porcja danych zostanie przesłana. Pozwala np. na wyświetlanie paska postępu.

Na tym etapie istotna będzie tylko funkcja *FileTransfer.upload()*, która wymaga szerszego opisania. Funkcja *FileTransfer.abort()* nie przyjmuje żadnych parametrów, a funkcja *FileTransfer.download()* przyda się w dalszej części artykułu i wtedy ją opiszemy.

Funkcja *FileTransfer.Upload()* przyjmuje w kolejności następujące parametry:
- **fileURL** – adres znajdującego się na urządzeniu pliku do załadowania na serwer,
- **server** – adres URL serwera, pod który będzie wysyłane żądanie, zakodowany funkcją **encodeURI()**,
- **success** – nazwa funkcji wywoływanej w przypadku powodzenia operacji,
- **error** – nazwa funkcji wywoływanej w przypadku niepowodzenia operacji,
- **options** – obiekt zawierający opcje przesyłanego żądania,
- **trustAllHosts** – wartość **false** lub **true**, określająca, czy aplikacja ma umożliwiać przesyłanie plików do dowolnych serwerów; domyślnie **false**.

Dużą rolę odgrywają liczne opcje, przekazywane w postaci obiektu klasy *FileUploadOptions*, jako parametr *options*. Oto one:
- **fileKey** – nazwa elementu zawierającego przesyłany plik, jaka zostanie przekazana w ramach wywołania HTTP do serwera,
- **fileName** – nazwa pliku, pod jakim ma być on zapisany na serwerze,
- **httpMethod** – rodzaj metody używanej do przesłania pliku; domyślnie POST, ale może być też PUT,
- **mimeType** – rodzaj treści przesyłanego pliku, podany zgodnie z tabelą 1. z trzeciej części niniejszego kursu (EP04/2015); domyślnie **image/jpeg**,
- **params** – obiekt zawierający opcjonalne pary nazw i wartości dodatkowych parametrów do przekazania do serwera w ramach żądania HTTP,
- **chunkedMode** – opcja umożliwiająca określenie, czy duże pliki mają być dzielone na części (domyślnie **true**),
- **headers** – tablica dodatkowych nagłówków (ich nazw i wartości), jakie mają być przesłane do serwera.

W naszym przypadku funkcję przesyłu plików na serwer będziemy wywoływać dwukrotnie – raz do przekazania zdjęcia, a raz do wysłania nagrania głosu. W obu przypadkach użyjemy metody POST. W przypadku powodzenia przesłania, do funkcji sukcesu przekazywany jest parametr w postaci obiektu *FileUploadResult*. Zawiera on cztery atrybuty:
- **bytesSent** – liczba bajtów przesłany do serwera,
- **responseCode** – kod odpowiedzi serwera (dostępne kody znalazły się w tabeli 3),
- **response** – treść właściwej odpowiedzi serwera,
- **headers** – nagłówki HTTP odpowiedzi serwera (aktualnie dostępne tylko w systemie iOS).

Jeśli odpowiedź była faktycznie pomyślna, to jej kod będzie wynosił 200, a interesująca nas treść, mówiąca o tym czy głos i zdjęcie użytkownika zostały pozytywnie zweryfikowane, będą w atrybucie *FileUploadResult.response*.

Pobieranie plików z serwera

W przypadku gdy użytkownik został pozytywnie zweryfikowany, nasza aplikacja pobiera spersonalizowany dla niego ekran powitalny i wyświetla go na ekranie. Treść ekranu mogłaby być zwracana bezpośrednio w odpowiedzi serwera na polecenie *FileTransfer.upload()*, ale może też zdarzyć się tak, że obraz powitalny znajduje się na zupełnie innym serwerze. Dla celów dydaktycznych przyjmiemy takie założenie i skorzystamy z funkcji *FileTransfer.download()*. Wymaga ona następujących parametrów:
- **source** – wywoływany adres, zakodowany za pomocą funkcji **encodeURI()**,
- **target** – miejsce zapisania pobieranego pliku na smartfonie,
- **success** – funkcja wywoływana w razie powodzenia operacji,
- **error** – funkcja wywoływana w razie niepowodzenia operacji,
- **trustAllHosts** – wartość **false** lub **true**, określająca, czy aplikacja ma umożliwiać przesyłanie plików do dowolnych serwerów; domyślnie **false**,
- **options** – dodatkowy, opcjonalny parametr, pozwalający na dosłanie dodatkowych nagłówków, w tym np. informacji o autoryzacji, jeśli taka jest wymagana.

W przypadku powodzenia, funkcja sukcesu otrzymuje jako parametr obiekt klasy *FileEntry*, który zawiera metodę *toURL()*, umożliwiającą uzyskanie lokalnego adresu pobranego pliku.

Synteza głosu

Na tym moglibyśmy zakończyć, ale obiecaliśmy, że wszystkie polecenia wydawane użytkownikowi, będą przekazywane głosowo. Aby uniknąć konieczności nagrywania i przechowywania plików audio dla każdej możliwej kombinacji zdarzeń i osób, skorzystamy z syntezy mowy, z użyciem wtyczki **org.apache.cordova.speech.speechsynthesis**.

Dokumentacja wybranej wtyczki jest bardzo skąpa, ale podstawowa obsługa syntezatora nie jest zbyt skomplikowana. Kluczowe jest stworzenie obiektu klasy *SpeechSynthesisUtterance* i przypisanie odpowiednich wartości jego atrybutom. My wykorzystujemy tylko cztery: *SpeechSynthesisUtterance.text*, w którym zapisujemy treść komunikatu do wypowiedzenia i *SpeechSynthesisUtterance.lang*, w którym określamy język komunikatu. Pozostałe dwa to funkcje, które mają się wykonać po rozpoczęciu syntezy mowy lub po jej zakończeniu (*SpeechSynthesisUtterance.start* i *SpeechSynthesisUtterance.end*) – przypisujemy im nazwy odpowiednich funkcji. Polecenie syntezy mowy wydajemy za pomocą funkcji *speechSynthesis.speak()*, której parametrem jest stworzony wcześniej obiekt klasy *SpeechSynthesisUtterance*.

Właściwy kod

Kod JavaScript realizujący opisaną procedurę, z użyciem omówionych funkcji, przedstawiono na **listingu 1**. Kod HTML znalazł się na **listingu 2**. Program, po naciśnięciu jedynego, zdefiniowanego przycisku, kolejno uruchamia funkcje: *app.scanBarcode()*, *app.takePhoto()*, *app.havePhoto()*, *app.recordVoice()*, *app.sendFiles()*, *app.getFiles()* i *app.welcome()*. Ponieważ ich działanie jest asynchroniczne, są one wywoływane w momencie zakończenia działania poprzednich z nich. Co więcej, w razie wystąpienia jakiegoś błędu, proces jest wstrzymywany i aplikacja wyświetla informację o błędzie, po czym ponownie można rozpocząć uwierzytelnianie użytkownika. Wymienione funkcje korzystają też z funkcji *app.speak()*, która po polsku czyta zadany tekst i uruchamia kolejną, wskazaną funkcję.

Funkcja *app.scanBarcode()* uruchamia polecenie skanowania kodu kreskowego. W przypadku jego wykrycia sprawdza, czy jest to kod QR (format *QR_CODE*), a jeśli tak, testuje, czy wczytana z kodu wartość jest większa niż 1000. Naturalnie w tym miejscu można by było testować identyfikator pod dowolnym innym kątem. Jeśli identyfikator wydaje się być poprawny, wczytany numer jest zapisywany, oraz wydawana jest komenda syntezująca mowę z poleceniem wykonania zdjęcia twarzy, która gdy tylko zacznie być wykonywana, włącza funkcję *app.takePhoto()*.

Funkcja *app.takePhoto()* włącza polecenie wykonania pojedynczego zdjęcia, bez możliwości jego edycji i bez zapisywania do albumu ze zdjęciami. Wskazujemy też, że po pomyślnym wykonaniu zdjęcia, ma się uruchomić funkcja *app.havePhoto()*.

Funkcja *app.havePhoto()* zapisuje adres zrobionego zdjęcia oraz uruchamia polecenie syntezy mowy, by

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 87542, pass: o8v5gec9

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP:
ftp://ep.com.pl, user: 87542, pass: o8v5gec9

Listing 1. Kod JavaScript programu z cz. 7 kursu

```
var app = {
    initialize: function() {
        this.bindEvents();
    },
    bindEvents: function() {
        document.addEventListener('deviceready', this.onDeviceReady, false);
    },
    onDeviceReady: function() {
        app.receivedEvent('deviceready');
        app.assignButtons();
    },
    assignButtons: function() {
        $(document).ready(function() {
            $('#scanButton').click(function() {
                app.speak("Proszę zeskanować kod",app.scanBarcode,null);
            });
        });
    },
    scanBarcode: function(){
        cordova.plugins.barcodeScanner.scan(
            function (result) {
                if (result.format=="QR_CODE")
                {
                    if (result.text>1000){
                        app.code=result.text;
                        app.speak("Proszę wykonać zdjęcie twarzy",app.takePhoto,null);
                    } else alert("Błędny identyfikator");
                } else alert("Błędny format kodu");
            },app.error
        );
    },
    takePhoto: function(){
        navigator.camera.getPicture(
            app.havePhoto,
            app.error,
            {
                allowEdit:false,
                mediaType:0,
                saveToPhotoAlbum:false
            }
        );
    },
    havePhoto: function(imageURI){
        app.face=imageURI;
        app.speak("Proszę podać imię i nazwisko",null,app.recordVoice);
    },
    recordVoice: function(){
        var src="/storage/emulated/0/tmprecording.3gp";
        var mediaRec = new Media(src,
            function() {
                app.voice=src;
                app.sendFiles();
            },app.error
        );
        mediaRec.startRecord();
        setTimeout(function() {
            mediaRec.stopRecord();
        }, 4000);
    },
    sendFiles:function(){
        app.speak("Dziękuję. Trwa weryfikacja",null,null);
        var serverAddress = "http://192.168.0.192/users/"+app.code+"/";
        var sFace=serverAddress+"face.php";
        var sVoice=serverAddress+"voice.php";
        var opt1 = new FileUploadOptions();
        opt1.fileKey = "file";
        opt1.fileName = "face.jpg";
        opt1.mimeType = "image/jpeg";
        var ft1 = new FileTransfer();
        var opt2 = new FileUploadOptions();
        opt2.fileKey = "file";
        opt2.fileName = "voice.mp3";
        opt2.mimeType = "audio/mpeg3";
        var ft2 = new FileTransfer();
        ft1.upload(app.face, encodeURI(sFace), function(r){
            if ((r.responseCode==200)&&(r.response=="OK"))
            ft2.upload(app.voice, encodeURI(sVoice), function(r2){
                if ((r2.responseCode==200)&&(r2.response=="OK"))
                app.getFiles();
            }, app.error, opt2);
        }, app.error, opt1);
    },
    getFiles:function(){
        var fileTransfer = new FileTransfer();
        var uri = encodeURI("http://192.168.0.192/userdata/"+app.code+"/welcome.jpg");
        var fileURL=cordova.file.dataDirectory+"welcome.jpg";
        fileTransfer.download(
            uri,fileURL,
            function(entry) {
                console.log("download complete: " + entry.toURL());
                app.welcome(entry.toURL());
            },app.error,true);
    },
    welcome:function(image){
        app.speak("Witamy "+ app.code,null,null);
        $("#welcome").attr('SRC',image);
        $("#welcome").css("width","100%");
        $("DIV.app").css("padding", 0);
    },
    speak: function(text,start,end){
        var u = new SpeechSynthesisUtterance();
```

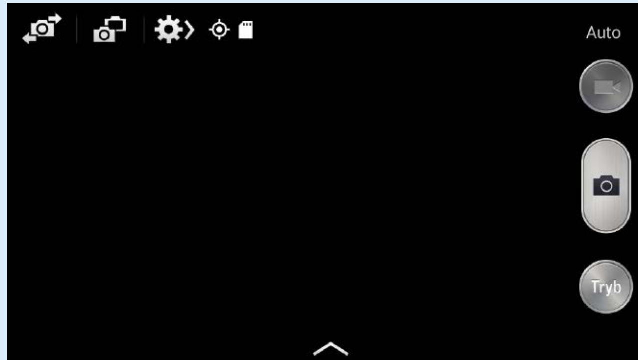
```
Listing 1. c.d.
        u.text = text;
        u.lang = ‚pl’;
        if (start!=null) u.start = start();
        if (end!=null) u.end = end();
        speechSynthesis.speak(u);
    },
    error: function(err){
        alert(„Wystąpił błąd „ + err);
    },
    code:null,
    face:null,
    voice:null,
    receivedEvent: function(id) {
        var parentElement = document.getElementById(id);
        var listeningElement = parentElement.querySelector(‚.listening’);
        var receivedElement = parentElement.querySelector(‚.received’);
        listeningElement.setAttribute(‚style’, ‚display:none;’);
        receivedElement.setAttribute(‚style’, ‚display:block;’);
        console.log(‚Received Event: ‚ + id);
    }
};
app.initialize();
```

```
Listing 2. Kod HTML powstałego programu
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv=”Content-Type” content=”text/html; charset=UTF-8” />
        <meta http-equiv=”Content-Security-Policy” content=”default-src ‚self’ data: gap: https://
ssl.gstatic.com ‚unsafe-eval’; style-src ‚self’ ‚unsafe-inline’; media-src *”>
        <meta name=”format-detection” content=”telephone=no”>
        <meta name=”msapplication-tap-highlight” content=”no”>
        <meta name=”viewport” content=”user-scalable=no, initial-scale=1, maximum-scale=1, minimum-
scale=1, width=device-width”>
        <link rel=”stylesheet” type=”text/css” href=”css/index.css”>
        <title>Hello World</title>
    </head>
    <body>
        <div class=”app”>
        <IMG id=”welcome” SRC=”” style=”width:0%”);
            <h1>Apache Cordova</h1>
            <div id=”deviceready” class=”blink”>
                <p class=”event listening”>Connecting to Device</p>
                <p class=”event received”>Device is Ready</p>
            </div>
        </div>
        <script type=”text/javascript” src=”cordova.js”></script>
        <script type=”text/javascript” src=”js/jquery-2.1.3.min.js”></script>
        <script type=”text/javascript” src=”js/index.js”></script>
        <div style=”display:table;width:100%;height:100px;background-color:green;font-size:xx-
large;text-align:center”>
            <div id=”scanButton” style=”display:table-cell; vertical-align: middle;”>SKANUJ</div>
        </div>
    </body>
</html>
```

nakazać użytkownikowi nagranie jego głosu z użyciem funkcji *app.recordVoice()*.

Funkcja *app.recordVoice()* określa miejsce lokalizacji pliku z nagranym głosem oraz tworzy nowy obiekt klasy *Media*, wskazując przy tym, że po poprawnym nagraniu ma się uruchomić funkcja *app.sendFiles()*. Następnie uruchamiana jest funkcja *Media.startRecording()* oraz za pomocą funkcji *setTimeout()* zlecane jest wykonanie funkcji *Media.stopRecording()* po 4 sekundach. Tu pojawia się pewien problem. Obiekt *Media* nie pozwala na przypisanie funkcji, która będzie się uruchamiała po faktycznym rozpoczęciu nagrywania (funkcja *Media.mediaStatus* działa w tym zakresie różnie, zależnie od platformy), a trzeba mieć na uwadze, że działa on asynchronicznie. Oznacza to, że polecenie *Media.stopRecording()* zostaje uruchomione po 4 sekundach od wydania polecenia *Media.startRecording()*, co wcale nie musi być równoznaczne z faktycznym rozpoczęciem nagrywania. Na wolniejszych urządzeniach i w zależności od stopnia aktualnego obciążenia systemu, faktyczne rozpoczęcie nagrywania może trochę potrwać.

Po wykonaniu nagrania uruchamia się funkcja *app.sendFiles()*, która wysyła dwa żądania HTTP do serwera o ustalonym adresie. Wraz z jednym żądaniem przesyłana jest wykonana wcześniej fotografia, a z drugim – nagranie dźwięku. Dla ułatwienia pracy serwera, adres wywołań zawiera w sobie zeskanowany wcześniej identyfikator użytkownika, dzięki czemu serwer wie, z którymi wzorcami ma porównywać otrzymane pliki. Przesył plików wykonywany jest pojedynczo. Najpierw wysyłamy fotografię i jeśli zostanie ona poprawnie zweryfikowana (kod odpowiedzi serwera 200, odpowiedź „OK”), przesyłane jest nagranie audio. Jeśli ponownie otrzymamy

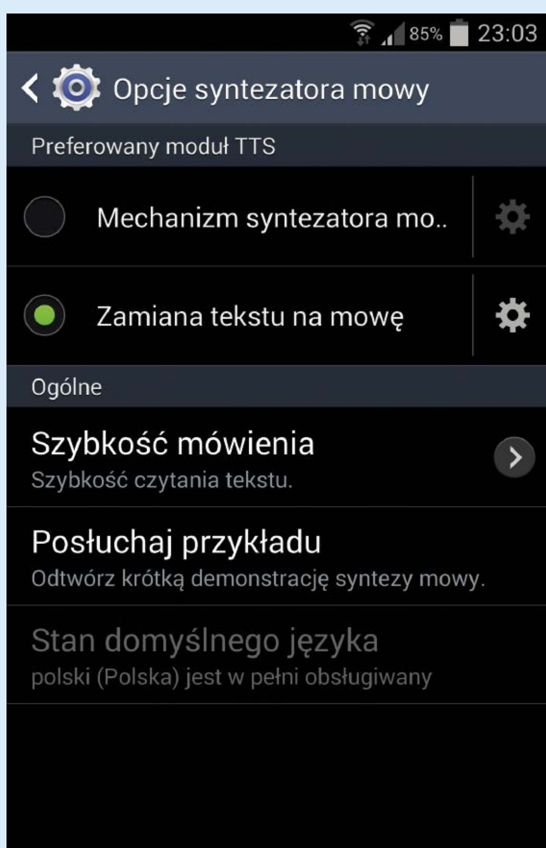


Rysunek 2. W trakcie wykonywania zdjęcia z użyciem pluginu *cordova-plugin-camera*, na ekranie pojawiają się liczne przyciski, umożliwiające użytkownikowi zmianę trybu robienia zdjęć

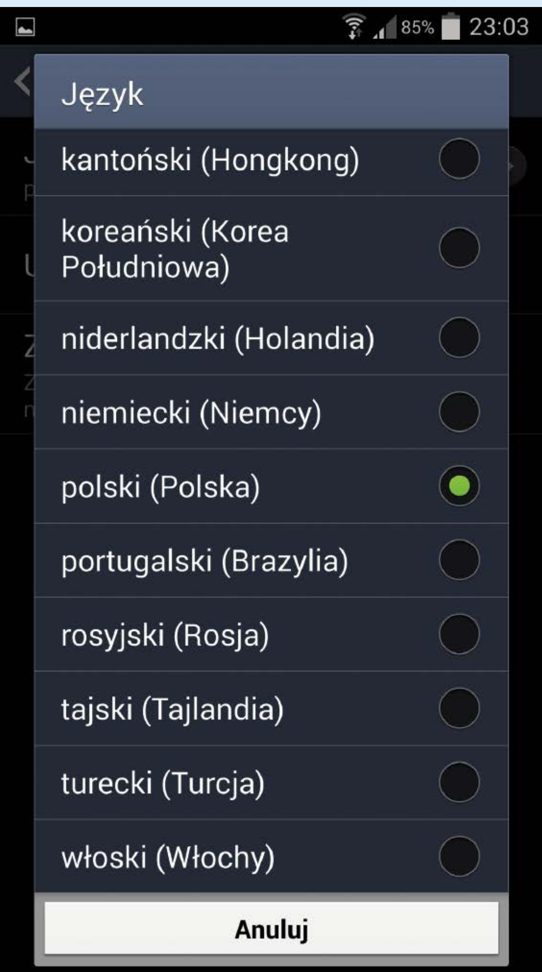
informację o poprawnej weryfikacji, urządzenie przyjmuje, że użytkownik jest autoryzowany i uruchamia funkcję *app.getFiles()*, której celem jest pobranie spersonalizowanego ekranu powitalnego dla danego użytkownika.

Działanie funkcji *app.getFiles()* jest dosyć proste. Łączy się ona z serwerem o podanym adresie by pobrać plik i zapisać go do wskazanej lokalizacji w pamięci urządzenia. Gdy to się powiedzie, uruchamiana jest funkcja *app.welcome()*, która wita użytkownika głosem, odnosząc się do jego numeru identyfikacyjnego oraz wyświetla jego ekran startowy, podmieniając plik graficzny na ten pobrany z serwera. Oczywiście, treść odczytywana na głos

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 87542, pass: o8v5gec9



Rysunek 3. Ustawienia syntezatora mowy w telefonie



Rysunek 4. Wybór języka głosu w jednym z modułów TTS telefonu urządzenia z Androidem

Tabela 3. Kody odpowiedzi serwerów HTTP

Kod	Nazwa
Odpowiedzi informacyjne	
100	Continue
101	Switching Protocol
Odpowiedzi pomyślne	
200	OK
201	Created
202	Accepted
203	Non-Authoritative Information
204	No Content
205	Reset Content
206	Partial Content
Przekierowania	
300	Multiple Choice
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect
Błędy klienta przesyłającego żądanie	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
405	Method Not Allowed
406	Not Acceptable
407	Proxy Authentication Required
408	Request Timeout
409	Conflict
410	Gone
411	Length Required
412	Precondition Failed
413	Request Entity Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
Błędy odpowiedzi serwera	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported

Nowości w platformie Cordova

Nowa wersja platformy Cordova, którą zainstalowaliśmy w tym kursie zawiera szereg zmian, a przede wszystkim usprawnień, które warto omówić w skrócie. Teoretycznie zmiany obejmują po prostu przede wszystkim podmianę używanych bibliotek na nowsze wersje, ale w praktyce wiąże się to z modyfikacjami, na które wypada zwrócić uwagę.

Zmienia się wygląd procesu kompilacji, który teraz przebiega nieco szybciej i jest bardziej zwięzły graficznie – nieistotne linijki z informacjami są czyszczone i pozostają tylko ważniejsze komunikaty, dzięki czemu całość jest trochę bardziej czytelna. Zmieniła się też lokalizacja gotowych plików. Od teraz są one umieszczane w katalogu wybranej platformy, w podkatalogu –w przypadku Androida – build\outputs\apk. Zmodyfikowano także ich domyślne nazwy – teraz noszą miano android-debug.apk i android-debug-unaligned.apk. Reszta zmian wydaje się nieistotna z punktu widzenia średnio-zaawansowanego użytkownika.

Warto tylko zwrócić uwagę na nowe repozytorium wtyczek Cordovy (**rysunek 5**). Jest ono o tyle mniej wygodne, że nie pozwala na wyszukiwanie po popularności pluginów.

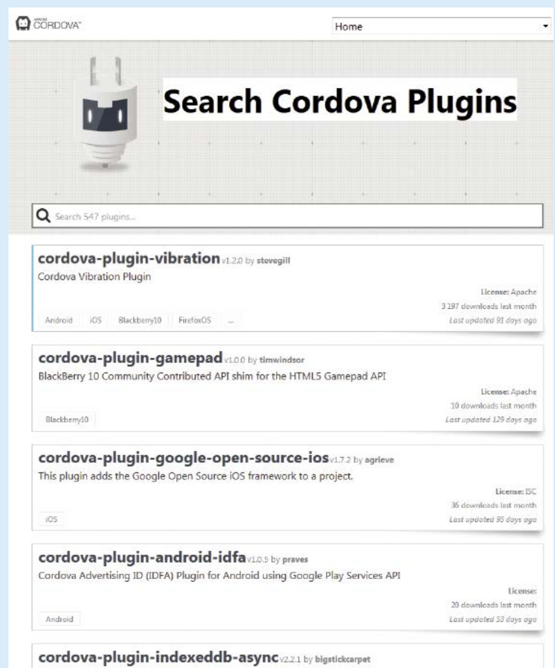
mogłaby być już pobierana z pierwszego z serwerów i brzmieć zupełnie dowolnie, ale dla prostoty przykładu i zwięzłości kodu nie wprowadzaliśmy takich dodatkowych opcji.

Warto jeszcze wspomnieć o przygotowanej przez nas funkcji *app.speak()*, która przyjmuje trzy parametry: treść komunikatu oraz odniesienia do funkcji uruchamianych po rozpoczęciu i po zakończeniu czytania wiadomości na głos. W funkcji tej na stałe ustawiamy język jako „pl", dzięki czemu jest on preferowanym językiem mechanizmu TTS (Text To Speech). Może się okazać, że obsługa polskiego języka wymaga zainstalowania go i skonfigurowania na telefonie. Jeśli tak, należy wejść do ustawień i poszukać kategorii „Język i wprowadzanie". Może ona się znajdować w dziale „Moje urządzenie". Wewnątrz wymienionej kategorii powinna się znaleźć pozycja „Opcje syntezatora mowy" (**rysunek 3**), w której ramach można wybrać mechanizm odpowiadający za czytanie tekstu na głos. Kliknięcie na ustawienia wybranego modułu TTS pozwala wybrać interesujący nas język (**rysunek 4**).

W standardowym projekcie pojawi się także jeszcze jeden problem, związany z kodowaniem znaków. Domyślnie nie obsługuje on języka polskiego, w efekcie czego, próba odczytania (czy nawet wyświetlenia) jakichkolwiek zdań z polskimi znakami diakrytycznymi poskutkuje wyświetleniem się tzw. krzaczków. Dlatego należy wprowadzić do pliku **index.html** linijkę informującą o zastosowaniu kodowania UTF-8 (**listing 2**).

Podsumowanie

W niniejszej części kursu pokazaliśmy, jak korzystać z kamery i mikrofonu w telefonie. Czytelnik powinien też samodzielnie być w stanie dojść do tego, jak skorzystać z zapisanej galerii i posługiwać się plikami multimedialnymi. Zademonstrowaliśmy również mechanizm wgrywania i pobierania plików za pomocą protokołu HTTP.



Rysunek 5. Nowe repozytorium wtyczek Cordovy, obsługiwane przez mechanizm *npm*

Zaprezentowaliśmy też sposób korzystania z kodów graficznych, które mogą być bardzo użyteczne w wielu aplikacjach. Całość uzupełnia obsługa mechanizmu TTS, czyli syntezatora mowy, która pozwala znacząco uatrakcyjnić działanie programu.

Przykładowy program ma jednak kilka mankamentów, które można by było usprawnić. Przykładowo dużym ograniczeniem mogą być ustawienia modułów do skanowania kodów kreskowych i robienia zdjęć, które w praktyce wymuszają użycie tylnej kamery i samodzielnie prezentują na ekranie komunikaty, które mogłyby być mylące dla użytkownika. Trzeba też mieć na uwadze, że skanowanie kodu QR czy kreskowego nie wprowadza praktycznie żadnego dodatkowego bezpieczeństwa, choć może być uzasadnione ze względu na obciążenie serwera, któremu znacznie łatwiej będzie rozpoznawać twarz konkretnej osoby, niż całej grupy upoważnionego personelu. Gdyby jednak chcieć wprowadzić dodatkowy mechanizm bezpieczeństwa, w oparciu o identyfikatory, można by się pokusić o zastosowanie czytnika znaczników RFID czy NFC, których podrobienie jest nieco trudniejsze niż kodu graficznego. Postaramy się pokazać, jak skorzystać z tych mechanizmów w jednym w kolejnych odcinków kursu.

Marcin Karbowniczek, EP

Zaprenumeruj na stronie avt.pl, e-mail: prenumerata@avt.pl
lub telefonicznie pod numerem: 22 257 84 22,
bieżący numer zamów na www.ulubionykiosk.pl

Poprzednie części kursu i dodatkowe materiały dostępne są na FTP: ftp://ep.com.pl, user: 87542, pass: o8v5gec9