PRZYGOTOWUJEMY APLIKACJĘ MOBILNĄ

Agnieszka Borowiecka, Katarzyna Olędzka
Ośrodek Edukacji Informatycznej i Zastosowań Komputerów
w Warszawie
{agnieszka.borowiecka, katarzyna.oledzka}@oeiizk.waw.pl

Abstract. All over the world mobile devices become more and more popular. Whereas knowing how to prepare a good application is crucial, in our proposition we use HTML, CCS, JavaScript and Cordova framework to prepare it. The programmer should combine different skills from designing an application, through coding HTML, CSS and JavaScript and compiling it into target formats. Such a process can allow not only to prepare a good product but also master programming competencies.

1. Wstęp

W świecie zdominowanym przez nowoczesne technologie i urządzenia mobilne nauczyciel powinien nie tylko inaczej zaplanować prowadzone przez siebie zajęcia szkolne, ale i przygotować dla uczniów takie materiały, które będą ciekawe i dostępne w formie elektronicznej. Przygotowanie zadań i materiałów rozszerzających omawiane zagadnienia w postaci aplikacji instalowanej na smartfonie lub tablecie może stanowić interesujące uzupełnienie dla standardowych podręczników. Także uczniowie są zainteresowani przygotowywaniem aplikacji na urządzenia mobilne i niejednokrotnie woleliby tym właśnie zajmować się na lekcjach informatyki. Czy oznacza to konieczność zagłębienia się w dość skomplikowane zagadnienia projektowania i programowania aplikacji na Androida, iOS czy Windows Phone? Instalację rozbudowanych środowisk programistycznych oraz np. Android SDK?

Proponujemy wybrać prostsze rozwiązanie. Wystarczy znajomość podstaw tworzenia stron internetowych za pomocą HTML i CSS, umiejętność przygotowywania prostych skryptów JavaScript oraz framework PhoneGap/Cordova.

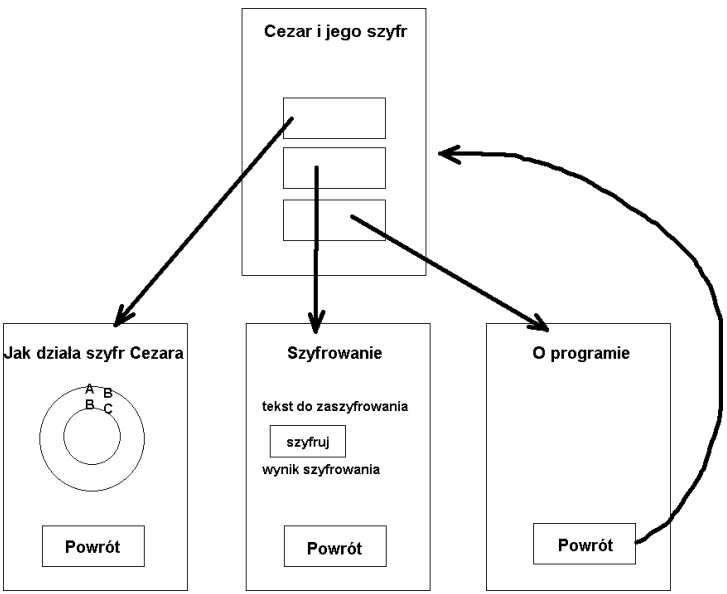
2. Projektowanie aplikacji

Wszystkie dobre aplikacje zaczynają się od pomysłu. Z jednej strony chcielibyśmy, by temat aplikacji był atrakcyjny dla uczniów, z drugiej chcemy „przemycić" trochę wiedzy algorytmicznej. W naszym artykule opiszemy przygotowywanie apli-

kacji na przykładzie zadania Szyfr Cezara. Ma on znaczenie historyczne i jest jednym z najpopularniejszych szyfrów podstawieniowych. Nazwa pochodzi od rzymskiego cesarza Juliusza Cezara, który szyfrował nim swoją korespondencję z Cyceronem. Można o tym szyfrze mówić z uczniami w zarówno szkole podstawowej, gimnazjum, jak i w szkole ponadgimnazjalnej.

Rysunek 1. Okno główne aplikacji Szyfr Cezara

Przygotowując mobilną aplikację pamiętamy, by prezentowane treści nie były zbyt rozbudowane. Ważne jest, by można było łatwo się z nimi zapoznać, nie przewijając wielokrotnie ekranu, czy wyszukując najważniejsze dane wśród gąszczu ogólnych informacji. Dodatkowo warto zadbać o atrakcyjność dodając interakcję.



Rysunek 2. Projekt aplikacji

Po ustaleniu, jakie dokładnie informacje zostaną przez nas wykorzystane, dzielimy je na poszczególne części, projektujemy system nawigacji, wyszukujemy i planujemy dodanie elementów wzbogacających aplikację – fragmentów kodu JavaScript, multimediów itp.

Planowana aplikacja będzie składała się z czterech ekranów:

- planszy startowej z przyciskami umożliwiającymi przejście do konkretnych części aplikacji,
- notatki wyjaśniającej działanie szyfru Cezara,
- formularza umożliwiającego zaszyfrowanie dowolnego podanego przez użytkownika tekstu,
- informacji o autorach aplikacji.

3. Zawartość aplikacji – przygotowujemy pliki HTML

Rozpoczniemy od przygotowania dokumentów HTML odpowiadających poszczególnym częściom aplikacji. Poza podzieleniem tekstu na cztery części i zapisaniem go w osobnych plikach, planujemy późniejsze formatowanie go za pomocą odpowiednich selektorów CSS, dodając do dokumentu dodatkowe znaczniki div, klasy (np. do wyświetlania linków jako przyciski), identyfikatory itp.

Przy tworzeniu aplikacji mobilnej warto zadbać, by wyglądała ona „dobrze" na różnych urządzeniach, była intuicyjna w obsłudze oraz zawierała wszystkie, niezbędne naszym zdaniem, treści. Najlepiej przygotować najpierw krótki, jedno lub dwustronicowy dokument HTML i wyświetlić go na kilku różnych urządzeniach, porównując np. czytelność tekstu, jakość i wielkość grafiki, łatwość korzystania z odnośników. Należy pamiętać, że konieczne jest precyzyjne zaplanowanie sposobu przechodzenia do kolejnych fragmentów treści, umożliwiające łatwy dostęp do nich również na niewielkich ekranach dotykowych.

Optymalizujemy tworzone strony do wyświetlania na urządzeniach mobilnych dodając odpowiedni znacznik **meta**:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Po przygotowaniu plików HTML wyświetlamy je w przeglądarce i sprawdzamy m.in. działanie odnośników, wyświetlanie grafiki itp. Warto obejrzeć przygotowane strony w różnych rozdzielczościach typowych dla urządzeń mobilnych, podejmując decyzję o tym, co należy zmienić w stosunku do standardowych ustawień. Zwracamy uwagę, że zgodnie z obowiązującą tendencją, oddzielamy zawartość merytoryczną aplikacji od jej wyglądu.

oprogramie.html (no symbol selected)

```
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Szyfr Cezara</title>
    <link href="css/styles.css" rel="stylesheet">

</head>
<body>

    <div class="container">
      <h1>O programie</h1>
      <p>Program przygotowano z wykorzystaniem: HTML5, CSS3, JavaScript
i frameworkiem Cordova/PhoneGap.</p>
      <p>Autorzy:</p>
      <ul>
          <li>Agnieszka Borowiecka</li>
          <li>Katarzyna Olędzka</li>
      </ul>
      <p>Ośrodek Edukacji Informatycznej i Zastosowań Komputerów w Warszawie</p>
      <a class="przycisk" href="index.html">Powrót</a>
    </div>

</body>
</html>
```

Rysunek 3. Przykładowa strona HTML

4. Wygląd aplikacji – elementy CSS

Podstawą jest czytelny interfejs, dlatego zadbajmy o odpowiednie powiększenie całego wyświetlanego tekstu.

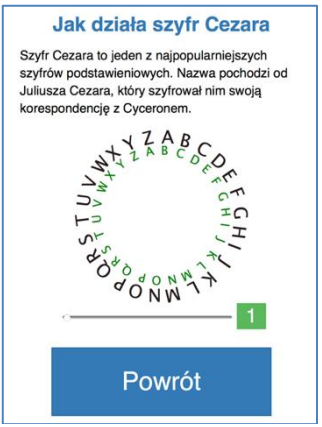
```
font-size: 1.3em;
line-height: 140%;
```

W kolejnym kroku zmieniamy odnośniki kierujące do poszczególnych części aplikacji tak, by łatwo było z nich korzystać na małym ekranie – nadajemy im wygląd dużych przycisków. Ponieważ nie wszystkie linki będziemy chcieli wyświetlać w identyczny sposób, definiujemy specjalną klasę **przycisk**.

```
/* wyswietlanie elementu jako przycisku */
.przycisk {
    width: 60%;
    padding: 40px;
    color: #fff;
    background-color: #337ab7;
    border: 2px #2e6da4 solid;
    font-size: 2em;
    text-align: center;
    text-decoration: none;
```

Rysunek 4. Fragment formatowania odnośników w formie przycisku

Możliwości CSS wykorzystamy także do animacji ilustrującej sposób dobierania w pary liter pierwotnego i zaszyfrowanego wyrazu. W tym celu wyświetlimy dwa przygotowane wcześniej rysunki przedstawiające kolejne litery alfabetu łacińskiego rozmieszczone na obwodzie koła.



Rysunek 5. Ilustracja działania szyfru Cezara

Jeden z rysunków stanowi tło dodatkowego znacznika **div**, drugi jest wyśrodkowany wewnątrz tego znacznika.

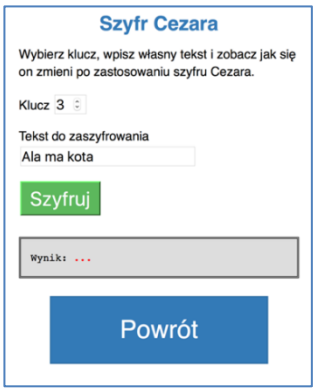
```
/* kola ilustrujace podstawianie liter*/
.kola {
    background-image: url("../images/kolo1.svg");
    background-repeat: no-repeat;
    background-position: center;
    margin-top: 20px;
}

#k2 {
    display: block;
    margin: 0 auto;
}
```

Rysunek 6. Formatowanie rysunków ilustrujących działanie szyfru

Kolejne miejsce, w którym możemy zastosować formatowanie za pomocą CSS to odpowiednie wyróżnienie zaszyfrowanego tekstu wyświetlanego na kolejnym ekranie naszej aplikacji.

Przygotowując plik CSS nie zapominamy o kolorach. Dobranie odpowiedniej palety barw jest niezmiernie ważne. W tym punkcie uczniowie będą mieli swoje preferencję, warto pozwolić im wykazać się własną inicjatywą. Jest to ciekawe zadanie. Mając dany szkielet aplikacji – kod HTML, zaprojektować wygląd aplikacji w istotnie różny sposób, modyfikując jedynie plik CSS.



Rysunek 7. Formatowanie rysunków ilustrujących działanie szyfru

5. Elementy interaktywne – JavaScript

W naszej aplikacji JavaScript wykorzystamy w dwóch miejscach – do szyfrowania podanego przez użytkownika tekstu zgodnie z wybranym kluczem oraz do sterowania animacją ilustrującą działanie szyfru Cezara. Przedstawimy szczegółowo pierwsze zagadnienie, jako bardziej istotne.

Zacznijmy od funkcji szyfrującej, której parametrami są znak i klucz, a wartością zaszyfrowany znak. Funkcja szyfruje małe litery alfabetu angielskiego przy pomocy klucza dodatniego. Najpierw sprawdzamy kod ASCII znaku, czy jest to mała litera. Jeśli tak, szyfrujemy znak, w przeciwnym przypadku pozostawiamy niezmieniony.

```
function szyfruj_znak(zn, klucz){
  if((zn >= 97) && (zn <= 122)){
    return((zn+klucz-97)%26+97);
  }
  else return(zn);
}
```

Funkcję możemy rozszerzyć o szyfrowanie cyfr i znaków diakrytycznych polskiego alfabetu, zadanie to pozostawiamy czytelnikowi.

```
function cezar(){
  var t="";
  var tekst=document.dane.tekst.value;
  var klucz=parseInt(document.dane.klucz.value);
  tekst=tekst.toLowerCase();
  for(i = 0; i < tekst.length; i++)
      t += String.fromCharCode(szyfruj_znak(
                  tekst.charCodeAt(i),klucz));
  document.getElementById("wynik").innerHTML = t;
}
```

Funkcja **cezar** pobiera tekst do zaszyfrowania z pola **tekst**, zaś klucz z pola **klucz** (jako zmienną typu integer). Następnie zamienia wielkie litery na małe, rozbija podany tekst na pojedyncze znaki, szyfruje je za pomocą funkcji **szyfruj_znak** i po połączeniu wpisuje w pole **wynik**.

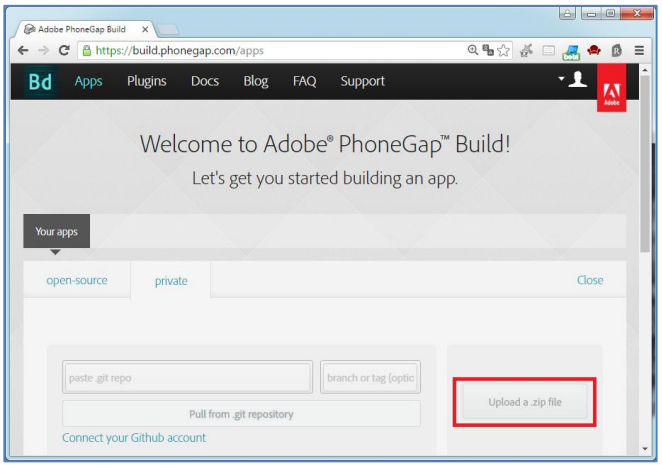
W ten sposób włączyliśmy elementy programistyczne do naszej aplikacji. Jest ona bardziej atrakcyjna dla użytkownika i bardziej wymagająca dla programisty. Szyfrujemy konkretny tekst, który w przypadku urządzeń mobilnych możemy komuś wysłać. Rozbudowując projekt możemy dodać odpowiednie opcje umożliwiające np. automatyczne wysyłanie wyniku szyfrowania.

Propozycji modyfikacji jest mnóstwo, można rozszerzyć szyfrowanie na wszystkie litery polskiego alfabetu, dodać odszyfrowywanie – nie powinno być to trudne, czy łamanie szyfru na podstawie częstości liter.

6. Od strony WWW do aplikacji

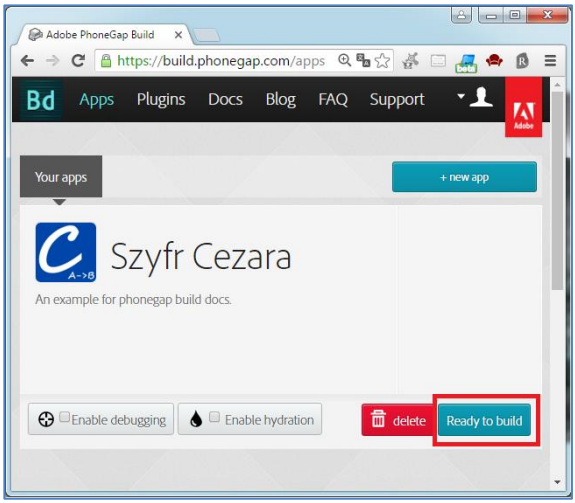
Jesteśmy gotowi do przygotowania aplikacji na urządzenie mobilne. W Internecie jest dostępnych wiele narzędzi darmowych, jak i płatnych. Skorzystamy z frameworku PhoneGap opartym na Cordovie. Za tego typu rozwiązaniem przemawia prostota. Korzystając z portalu firmy Adobe https://build.phonegap.com/, można zbudować aplikację bez instalacji potrzebnych narzędzi na naszym komputerze.

Najpierw tworzymy paczkę – wszystkie pliki html, css i javascript, które utworzyliśmy poprzednio, pakujemy w formacie zip. Pamiętajmy, że główny plik html powinien się nazywać **index.html**. Następnie tworzymy konto w serwisie i postępujemy zgodnie z instrukcją na ekranie. W zakładce prywatne aplikacje (**private**) wybieramy opcję **Upload a .zip file** i przesyłamy nasz plik zip.



Rysunek 8. Przysyłanie przygotowanych plików

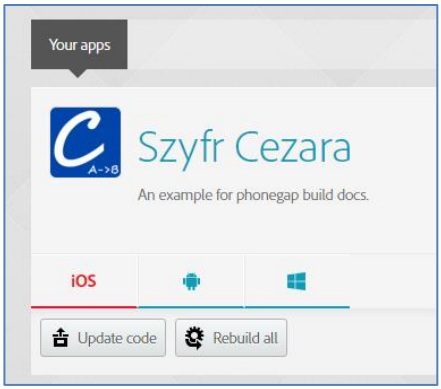
Po wgraniu pliku pojawi się strona z przyciskiem **Ready to build**. Kliknięcie w niego spowoduje wygenerowanie aplikacji gotowej do zainstalowania na urządzeniach mobilnych.



Rysunek 9. Generowanie aplikacji

Warto zauważyć, że opisujemy najprostszy sposób tworzenia aplikacji. W dokumentacji można przeczytać o podstawowych ustawieniach, np. jak dodać ikonki do programu, jakie parametry warto określić w pliku **config.xml** oraz jak podpisać aplikację (w przypadku systemu IOS jest to kluczowe!).

Po zakończeniu generowania aplikacji pojawi się okienko umożliwiające pobranie pliku instalacyjnego poprzez kliknięcie lub wczytanie QR kodu.



Rysunek 10. Pobieranie aplikacji

Tak przygotowaną aplikację pobieramy, instalujemy na urządzeniu mobilnym, testujemy, poprawiamy, i …

W opisanym przez nas rozwiązaniu korzystamy z portalu firmy Adobe. Możemy jednak pobrać potrzebne oprogramowanie – framework PhoneGap lub Cordova, zainstalować i generować aplikację w trybie offline. Jednorazowy proces instalowania jest żmudny, ale samo generowanie aplikacji nie jest już takie czasochłonne. Wymaga wydania kilku komend w trybie tekstowym. Zachęcamy do eksperymentów!