Přístupné Nette aplikace

Připravili: Radek Pavlíček, Roman Kabelka, TyfloCentrum Brno, o. p. s.
Určeno pro účastníky Poslední soboty v Brně, 30. 10. 2010

Jak nevidomý/slabozraký uživatel pracuje s PC?

Nevidomí a slabozrací lidé používají běžné počítače, vybavené tzv. **asistivní technologií**, která jim práci s počítačem umožňuje. V případě slabozrakých uživatelů se jedná o tzv. **softwarovou lupu**, což je program, který informace na obrazovce umí zvětšit a v případě hlasové podpory doplnit i čtením informací z obrazovky. Nevidomí uživatelé pak používají tzv. **screenreader** (odečítač/čtečku obrazovky), který v kombinaci s hlasovou syntézou umí informace z obrazovky přečíst, v případě připojení **braillského řádku** je pak může uživatel mít k dispozici i v Braillově slepeckém písmu.

Jaká jsou specifika práce nevidomých a slabozrakých uživatelů PC?

- **Nevidomý uživatel nepracuje s PC intuitivně, ale analyticky** – musí se naučit konkrétní postupy a řešení – zásadní rozdíl oproti práci slabozrakého uživatele či uživatele bez zrakové vady.
- **Nevidomý uživatel** musí mít **operační systém a některé aplikace speciálně nastaveny** tak, aby byly co nejlépe zpřístupněny.
- **Nevidomý uživatel obsluhuje PC** a veškeré programy výhradně **z klávesnice** pomocí klávesových povelů (zkratek).
- **Nevidomý uživatel** získává informace **lineárně** – **chybí mu kontext** zobrazované informace.
- **Nevidomému uživateli** jsou pomocí screenreaderu, neboli odečítače obrazovky, zpřístupněny pouze **informace v textové podobě** (není tedy schopen pracovat s obrázky, grafy, atd. – zde je potřeba použít tzv. taktilní grafiku).
- **Slabozraký uživatel** v důsledku použití softwarové lupy **vidí v jednu chvíli pouze (malou) část obrazovky**, chybí mu kontext zobrazených informací.
- Někteří **slabozrací uživatelé potřebují jinak nastavené barevné schéma obrazovky**, než je standardní nastavení (např. „Vysoký kontrast černá“).

Principy práce s webovou stránkou

Neexistuje jediný správný postup/přístup, nevidomí/slabozrací uživatelé kombinují různé způsoby práce s obsahem. Mezi nejčastěji používané patří:

- seznámení se se strukturou a obsahem stránky pomocí nadpisů,
- projití stránky tabulátorem po odkazech a dalších prvcích, které mohou získat focus,
- postupné čtení stránky *odshora dolů*,
- hledání požadované informace pomocí funkce Najít (Ctrl+F).

Protože neexistuje jediné správné řešení a nelze předvídat, jak uživatel k práci s webovou stránkou přistoupí, je vhodné, aby autor s tímto počítal a umožnil použití co nejvíce způsobů.

Přístupnost formulářů

Existují tři způsoby, jak může uživatel screenreaderu k formuláři přistupovat

- na jednotlivé formulářové prvky se bude přesouvat pomocí **tabulátoru**,
- pro práci s formulářem použije **kurzorové šipky**,
- **oba** výše uvedené **způsoby zkombinuje**.

Kdy je formulář přístupný?

- Je plně ovladatelný z klávesnice.
- Všechny formulářové prvky mají řádné přiřazeny relevantní popisky (*label, for, id*).
- V případě rozsáhlejších formulářů (či všude tam, kde to má smysl), jsou použity značky *fieldset* a *legend* pro rozdělení formuláře do menších logických celků.
- Informace o povinné položce je součástí relevatního popisku, případně je umístěna na začátku formuláře
- Fromulář má nápovědu k jednotlivým políčkům (tj. uživateli je jasné, co má kam zadat).
- Formulář má ošetřeny chyby na vstupu a uživatele o nich informuje přístupným způsobem.

CSS

Ačkoliv CSS specifikace obsahuje media type jako *aural, braille, embossed*, **nejsou** v praxi **asistivními technologiemi brány v potaz**. Proto se jimi nemusíte zabývat a stačí se soustředit na běžné media type (screen, all, print).

Javascript

Otázka přístupnosti Javascriptu není jednoduchá a zde se dá napsat asi jenom jedno – **testovat, testovat, testovat** ;-)

AJAX aplikace a přístupnost – kde mohou být problémy?

1. Problematické jsou situace, kdy **při události** (ať už nastane automaticky nebo jako výsledek interakce uživatele) není znovunačtena celá stránka, ale **novým obsahem je nahrazena jen její část**. Odečítač obrazovky je sice schopen změny identifikovat, ovšem neexistuje obecný postup, jak uživatele o těchto změnách informovat (odečítač neumí změněnou oblast implicitně pojmenovat), nehledě na to, že provedená změna může být nedůležitá a informace o ní může uživatele spíše obtěžovat, než mu být k užitku (příkladem takové nedůležité změny může být obnova reklamního banneru). V současnosti u nás běžně používané odečítače umí identifikovat a pojmenovat na základě titulku jen změny v rámech (*iframe*).

2. **Ovladatelnost z klávesnice.** Protože webová aplikace si žádá sofistikovanější ovládání, nevystačí si jen s klasickými odkazy a potřebuje i jiné elementy k zobrazení těchto ovládacích prvků. Ty ale často při své aktivaci vyvolávají události voláním skriptu. Odkazy tedy nevyvolávají odskok na cílové místo v pravém slova smyslu, což uživatele neseznámené s prací v takové aplikaci může značně zmást. Častější je ovšem využití

jiných elementů, jakými je například značka *div*, která je z hlediska skriptování nejtvárnější a která také mimo jiné postrádá jakoukoliv sémantickou informaci o svém obsahu. Události jsou potom asociovány s elementem pomocí atributu onclick, v horším případě onmouseover (událost přejetí myší nemá ekvivalent v ovládání z klávesnice). Fakt, že na element je navěšena událost při kliknutí (*onclick*), je schopen odečítač odhalit a nabídnout možnost simulace kliknutí na toto místo. Protože však takovému elementu není dán fokus v pravém slova smyslu, jde o metodu, jež nemusí fungovat stoprocentně. Z nemožnosti udělit elementu fokus vyplývá, že ovládací prvek tvořený elementem *div* či jiným, určeným pouze pro prezentaci, není zahrnut do tab order (klávesou tabulátor jsou ve všech prohlížečích dosažitelné pouze odkazy a formulářové prvky).

3. **Webová aplikace dělí své rozhraní na několik obsahově specifických sekcí** (navigační část, hlavní obsah, vyhledávání atp.), **pro něž neexistuje jednotný způsob vyznačení v kódu.** Dnes je již obvyklé používat systém navigačních odkazů na počátku stránky a uvozování jednotlivých sekcí pomocnými (skrytými) nadpisy, nicméně webová aplikace je často vyvíjena ve frameworcích a sestavována z widgetů, které nemají sémantickou stránku přizpůsobenu obsahu, což opět vychází z nasazení skriptování (masivní generování obsahu skripty).

Praktické rady pro tvorbu přístupných aplikací

Směrodatný je kód odesílaný do prohlížeče, jenž asistivní technologie (odečítač) zpracovávají, aby jej mohly prezentovat postiženému uživateli. Takový výstup je porovnatelný s pravidly a doporučeními pro přístupný web (W3C WCAG).

Vnitřní struktura Nette aplikace může napovědět, jak se vyhnout některým problémům s přístupností.

- **Přesvědčte se, že každý ovládací prvek aplikace (odkaz, tlačítko) posílající požadavek na server (akce presenteru, signál komponenty) má schopnost získat fokus - je začleněn do cyklu tab order (průchod tabulátorem) v prohlížeči.**
    - Automaticky mají schopnost získat fokus odkazy a formulářové prvky.
    - Pomocí atributu tabindex="0" dle WAI-ARIA může získat fokus jakýkoli element. Přesto používejte ovládací prvky z elementů a jen vyjímečně.
- **Přesvědčte se, že každý blok v šabloně, komponenty (včetně podkomponent) a snippety nekorespondují s částí stránky, o níž by uživatel měl vědět, kde začíná a kde končí.**
    - Nastylování sémanticky anonymních elementů a nestačí.
    - Části stránky nesoucí jeden druh informace či provádějící jednu činnost uvozujte řádně vyznačenými nadpisy. Na nevhodných místech lze použít styl odsunující text mimo viditelnou oblast. *.hidden { position: absolute; top: auto; left: -1024px; width: 1px; height: 1px; overflow: hidden; }*
    - Pomocí atributu role dle WAI-ARIA či příslušných HTML5 elementů lze určit účel dané části stránky (bloku, komponenty, snippetu) a zachovat tak logickou strukturu ze Nette šablony i ve výsledném HTML kódu. Přesto z důvodů zpětné kompatibility nezapomínejte na uvozování nadpisy.
    - Pokud komponenta obsahuje podkomponenty, měly by jejich nadpisy dodržovat hierarchii úrovní nadpisů.

    - Pokud se rozhodnete vyznačit obsah komponenty, která obsahuje podkomponenty (kontejner), jako seznam, neuvozujte je již podnadpisy. Každá položka seznamu jasně vymezuje začátek a konec podkomponenty (vhodné pro formuláře).
- **Přesvědčte se, že ajaxové požadavky dávají jasně dohledatelnou odezvu a to i během načítání.**
    - Uživatelé asistivních technologií vnímají najednou pouze prvek, jež má fokus nebo omezený úsek textu. Při vyvolání ajaxového požadavku zůstává fokus stále na jednom místě, proto uživatelé musí stránku projít a objevit změny.
    - Pomocí atributů *role, aria-live*, … dle WAI-ARIA lze dosáhnout toho, aby změny v invalidovaných komponentách či snippetech byly čteny adekvátním způsobem.
    - Ajaxový požadavek, jenž invaliduje komponentu nebo snippet jednoznačně závislý na další interakci, by měl zajistit i přesun fokusu do oblasti komponenty/snippetu na první fokusovatelný ovládací prvek v něm. Neobsahuje-li část stránky přímo závislá na ajaxovém požadavku žádný ovládací prvek, je vhodné použít atribut aria-live dle WAI-ARIA.

Odkazy na další zdroje (http://jdem.cz/hr8n7)

- www.poslepu.cz
- www.blindfriendly.cz
- http://zdrojak.root.cz/serialy/pristupnost-dynamickych-webovych-aplikaci/
- http://www.w3.org/WAI/intro/aria.php
- http://webaim.org/blog/future-web-accessibility-updates/