Kamil Zasada

Wprowadzenie do testowania aplikacji AngularJS

Miniony rok, tudzież rok poprzedni, były niewątpliwie dobrym okresem języka JavaScript. Przez wiele lat JavaScript nie był uznawany za pełnoprawny język programowania, który mógłby posłużyć do budowania skomplikowanych aplikacji internetowych. Sytuacja uległa zmianie, gdy na scenę zaczęły wkraczać frameworki MVC oraz wraz z upowszechnianiem się idei budowania aplikacji typu SPA. Aktualnie wśród frameworków MVC prym wiedzie AngularJS. W niniejszym artykule przedstawione zostaną sposoby testowania aplikacji bazujących na AngularJS.

ANGULARJS

AngularJS jest frameworkiem służącym do budowania aplikacji internetowych typu SPA (ang. *Single Page Application*) działających w środowisku przeglądarki internetowej. Nazwa tego typu aplikacji wzięła się stąd, że początkowo pobierana jest tylko jedna strona (`index.html`), a reszta budowana jest dynamicznie w zależności od stanu aplikacji, poprzez pobieranie, za pomocą żądań XHR, kolejnych fragmentów strony zapisanych w odrębnych plikach HTML.

AngularJS zaadoptował wzorzec MVC (ang. *Model View Controller*) na swój własny sposób, korzysta z mechanizmu wstrzykiwania zależności (ang. *Dependency Injection*), a komponenty można agregować w moduły, dzięki czemu aplikacje napisane w AngularJS posiadają czytelną strukturę i są skalowalne. W warstwie widoku wykorzystywany jest HTML jako język szablonów, który w połączeniu z wbudowanymi dyrektywami stanowi doskonałe narzędzie do przygotowywania stron aplikacji. Logika biznesowa tworzona jest przy użyciu języka JavaScript i komponentów AngularJS, a wygląd aplikacji definiowany jest przy użyciu stylów CSS.

Framework od samego początku tworzony był z myślą o tym, aby aplikacje pisane z jego wykorzystaniem były w pełni testowalne. Udostępnia programiście mechanizm wstrzykiwania zależności umożliwiający podział aplikacji na małe fragmenty mogące działać w całkowitej separacji, co wpływa na łatwość pisania testów jednostkowych.

Kod źródłowy AngularJS jest otwarty, dobrze udokumentowany i pokryty testami jednostkowymi, dzięki czemu o mechanizmach działania frameworka można dużo się dowiedzieć, czytając sam kod. Dokumentacja API dostępna pod adresem http://docs.angularjs.org zawiera również przykłady testów E2E (ang. *End-to-End*). Daje to poczucie, że zespół Google dba o jakość swojego produktu.

Zespół programistów Google odpowiedzialny za rozwój frameworka przygotował również narzędzia, które ułatwiają pisanie i uruchamianie testów, takie jak: Karma (http://karma-runner.github.io/0.12/index.html), Protractor (http://angular.github.io/protractor/) oraz moduł ngMock udostępniający funkcje do wstrzykiwania zależności w testach jednostkowych oraz atrapy (ang. *mocks*) wbudowanych serwisów, np. `$httpBackend` wykorzystywany w aplikacji przez serwis `$http` do wykonywania żądań XHR.

Wybierając AngularJS jako bazę dla przyszłego projektu, otrzymujemy kompletny zestaw narzędzi do przygotowania aplikacji o wysokiej jakości kodu – kodu dobrze udokumentowanego i łatwego w utrzymaniu. Wszystkie te czynniki sprawiają, że AngularJS jest obecnie najpopularniejszym frameworkiem do tworzenia aplikacji JavaScript.

TESTY AUTOMATYCZNE

JavaScript jest językiem dynamicznie typowanym, interpretowanym, a nie kompilowanym, przez co brak jakichkolwiek narzędzi do sprawdzania poprawności działania aplikacji przed jej uruchomieniem. Istnieją narzędzia do przeprowadzania statycznej analizy kodu, np. JSHint, JSLint, jednakże pozwalają one jedynie zminimalizować ryzyko wystąpienia błędu spowodowanego błędami w składni kodu źródłowego. Większość błędów pojawia się w czasie działania aplikacji, dlatego tak ważne jest przeprowadzanie testów dynamicznych.

W wielu projektach do przeprowadzania testów działania aplikacji powoływany jest specjalny zespół testerów. Wykonują oni zarówno testy automatyczne, jak i manualne. Tego typu specjalizacja jest jak najbardziej potrzebna, szczególnie w dużych projektach, jednakże spora część testów aplikacji powinna być wykonywana już we wczesnych stadiach rozwoju – wykonywana bezpośrednio przez programistów. Testy automatyczne pisane równocześnie wraz z kodem aplikacji lub przed (w podejściu *Test-Driven Development*) przyczyniają się do tego, że aplikacja jest lepiej zaprojektowana, wcześniej można wykryć ewentualne błędy w logice działania procesów wewnętrznych, a późniejsze zmiany w kodzie źródłowym (ang. *code refactoring*) są mniej bolesne. Jeśli aplikacja jest w pełni pokryta testami, to przed opublikowaniem kolejnej wersji wszystkie obszary aplikacji zostaną gruntownie przetestowane. Ma to szczególne znaczenie w momencie aktualizacji bibliotek pochodzących od zewnętrznych dostawców, np. AngularJS czy jQuery.

W niniejszym artykule skupię się na przedstawieniu narzędzi i sposobów pisania automatycznych testów jednostkowych oraz testów end-to-end dla aplikacji bazującej na AngularJS.

BDD, czyli Behavior-Driven Development

Mówiąc o tematyce pisania testów, nie sposób nie wspomnieć o metodyce *Behavior-Driven Development*. Jest ona rozwinięciem innej metodyki programowania – *Test-Driven Devlopment*. W podejściu TDD najpierw piszemy testy jednostkowe dla danego fragmentu aplikacji. Ponieważ brak jeszcze właściwego kodu aplikacji, testy te po uruchomieniu będą kończyły się niepowodzeniem. Dlatego w drugim etapie programista przygotowuje kod aplikacji. Testy określają specyfikę działania funkcji, modułu, a kod źródłowy realizuje opisane wymagania. Na tym etapie testy kończą się poprawnie i programista może przejść do ostatniego etapu, czyli porządkowania kodu. Zazwyczaj kod, który powstanie w drugim etapie, spełnia wymagania, jednakże może on wymagać poprawek zwiększających czytelność zaimplementowanej logiki czy zastosowania rozwiązań poprawiających wydajność aplikacji. Tak naprawdę etap porządkowania kodu nie jest ostatnim etapem. W rzeczywistości poprawa kodu źródłowego, zmiana specyfikacji testów to proces ciągły. Ze względu na pokrycie kodu aplikacji testami proces jego ulepszania jest mniej ryzykowny.

Pisanie testów jednostkowych to tak naprawdę definiowanie zachowania kolejnych fragmentów aplikacji, dlatego możemy rozszerzyć termin TDD i mówić o *Behavior-Driven Development*. Zachowanie aplikacji definiują nam nasze testy, które stanowią doskonałą dokumentację systemu – dokumentację zawsze aktualną.

Testy jednostkowe

Testy jednostkowe (ang. *unit tests*) to pierwszy krok na drodze do wprowadzenia metodyki *Test-Driven Development*. Skupiają się one na testowaniu pojedynczych, możliwie jak najmniejszych fragmentów aplikacji. Testy opisują logikę biznesową aplikacji, ale tylko na bardzo niskim poziomie – poziomie pojedynczych funkcji. Zakładają one testowanie pojedynczych bloków programu w całkowitym odseparowaniu od reszty modułów, dlatego tak ważne jest minimalizowanie odpowiedzialności poszczególnych modułów. Przykładowo, funkcja odpowiedzialna za pobieranie danych z serwera poprzez RESTful API, modyfikowanie drzewa DOM, zapisująca stan aplikacji z localStorage itd. z pewnością realizuje zbyt dużo zadań i należałoby rozważyć jej podział na mniejsze fragmenty, których testowanie będzie zdecydowanie łatwiejsze.

Aplikacja powinna być rozważana jako połączenie modułów wzajemnie ze sobą współpracujących. Zmiana wewnętrznych mechanizmów działania pojedynczego modułu nie powinna mieć wpływu na działanie modułów od niego zależnych. Moduły powinny być rozpatrywane jako czarne skrzynki – ważna jest poprawność danych wejściowych i wyniku. Testy jednostkowe skupiają się właśnie na testowaniu tych czarnych skrzynek pod kątem poprawności działania ich wewnętrznych funkcji oraz udostępnianego interfejsu. Nie należy ich mylić z testami integracyjnymi (np. testy end-to-end) sprawdzającymi poprawność współpracy pomiędzy modułami. Jeśli interfejs udostępniany przez moduł ulegnie zmianie, to testy jednostkowe taką zmianę powinny wykryć.

Zaleta testów jednostkowych to nie tylko wykrywanie błędów na wczesnym etapie powstawania aplikacji, ale również inne korzyści. Wymuszają one przykładanie większej staranności na etapie projektowania aplikacji – podziału kodu na logiczne jednostki odpowiedzialne za wykonywanie tylko ściśle określonych zadań. Pisząc w pierwszej kolejności testy, bardziej zastanawiamy się nad strukturą kodu, samą logiką biznesową, dzięki czemu cały system będzie lepiej zaprojektowany, a wszelkie niejasności w logice mogą zostać rozwiązane przed tym, jak oprogramowany zostanie cały komponent. Proces porządkowania kodu dzięki pokryciu testami jest mniej podatny na błędy, dzięki czemu łatwiej jest wprowadzać poprawki związanie np. z poprawą wydajności. Oprócz tego testy jednostkowe stanowią znakomitą dokumentację naszego systemu.

Testy End-to-End

Testy jednostkowe są pierwszą linią obrony przed błędami w aplikacji. Niestety specyfika tych testów polegająca na tym, że testy wykonywane są w odseparowaniu od reszty komponentów, wpływa na to, że nie wszystko jesteśmy w stanie przetestować za ich pomocą. W szczególności utrudnione jest wykonanie testów integracyjnych, których zadaniem jest sprawdzenie poprawności współpracy pomiędzy różnymi komponentami.

Rozwiązaniem tego problemu jest skorzystanie z testów end-to-end. Nie tylko pozwalają nam przeprowadzić testy integracyjne, ale również zmniejszyć czas potrzebny na testy manualne. Manualne retesty oraz testy kolejnych funkcji systemu stanowią dość duży procent całkowitego czasu tworzenia aplikacji. Automatyzacja poprzez testy end-to-end zmniejsza te proporcje, dzięki czemu więcej czasu zostaje na implementację nowych pomysłów.

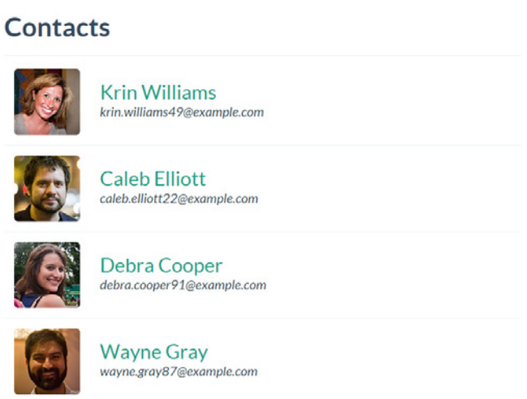
APLIKACJA

Nieodłączną częścią artykułu jest przykładowa aplikacja wykorzystująca narzędzia i koncepcje testowania przedstawione w dalszej części. Kod źródłowy aplikacji dostępny jest pod adresem:
https://github.com/zkamil/programistamag.angular.tdd.

Aplikacja to uproszczona lista kontaktów. Składa się z dwóch widoków:

- lista wszystkich kontaktów (patrz: Rysunek 1)
- szczegóły kontaktu wybranego poprzez kliknięcie na pojedynczy kontakt (patrz: Rysunek 2)

Dane prezentowane w aplikacji pochodzą z Random User Generator (https://randomuser.me/). Informacje na temat tego, jak uruchomić aplikację, znajdują się w dalszej części artykułu. Kod źródłowy aplikacji znajduje się w katalogu /src/app, testy jednostkowe w katalogu /test/unit, natomiast testy end-to-end w katalogu /test/e2e. W katalogach z testami korzystam z konwencji polegającej na odzwierciedleniu struktury katalogów aplikacji. Uważam ją za najbardziej czytelną.



Rysunek 1. Przykładowa aplikacja – lista wszystkich kontaktów

Rysunek 2. Przykładowa aplikacja – widok pojedynczego kontaktu