**9. Projekt indywidualny - tydzień 1 z 4**

**Wyzwania:**

* przeanalizujesz projekt graficzny,
* zaplanujesz podział strony na komponenty i sekcje,
* stworzysz styleguide,
* rozpoczniesz tworzenie layoutu i podstawowych elementów.

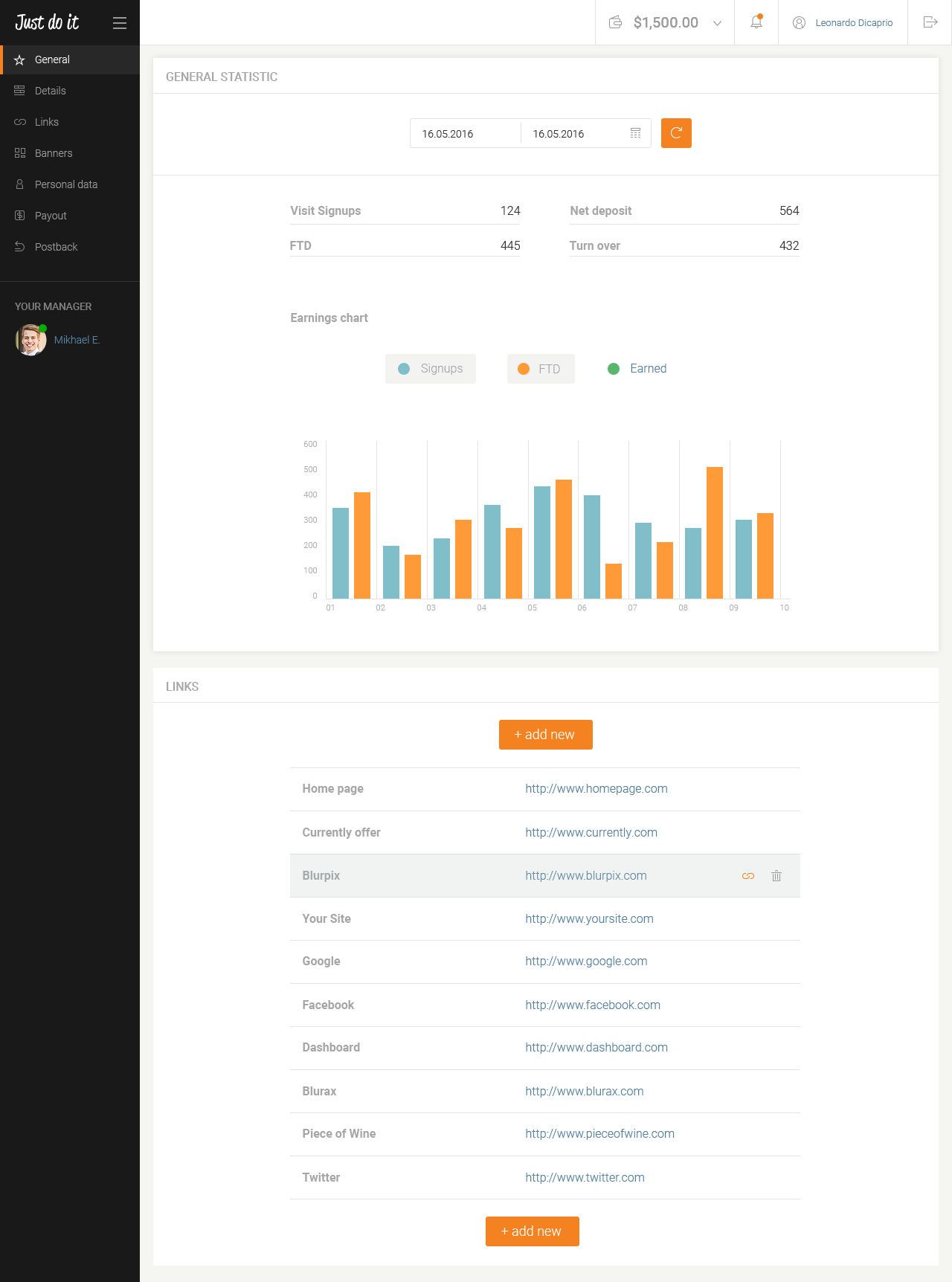
**Wstęp**

Osiem tygodni wprowadzenia już za Tobą — czas na większy projekt! Bedziesz nad nim pracować przez następne 4 tygodnie. W każdym tygodniu dostaniesz ciekawe informacje dotyczące poprawienia jakości Twojej pracy, jednak najważniejsze rzeczy znajdą się w tym module: dowiesz się, jak rozpocząć pracę nad projektem, oraz jakim podejściem warto się kierować przy tego typu zadaniach.

Projekt został podzielony na etapy realizacji. Pierwszy etap jest obowiązkowy do zaliczenia projektu, ale gorąco zachęcamy do wykonania jak największej części projektu.

**9.1. Specyfikacja projektu**

Twoim zadaniem jest przygotowanie widoku panelu administracyjnego do systemu zarządzania bannerami i linkami. Poniżej znajdziesz specyfikację poszczególnych etapów realizacji, ale nie rozpoczynaj pracy dopóki nie przeczytasz całego modułu. Znajdziesz w nim wiele informacji, które pomogą Ci rozpocząć prace nad projektem i uniknąć wielu błędów.

[](https://uploads.kodilla.com/bootcamp/wdp/WDP-1-1-0.png)

**Pliki graficzne**

Zaczynamy od pobrania [plików projektu](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1.zip).

Po rozpakowaniu pliku zip, znajdziesz w nim:

* plik guide.psd w którym znajdziesz przykłady komponentów używanych w projekcie (może się przydać na początku projektu),
* plik dashboard.psd w którym znajdziesz wszystkie widoki naszego projektu,
* katalog preview w którym znajdują się zapisane pliki .png każdego widoku,
* katalog icons w którym znajdują się wykorzystane w projekcie ikony w formacie .svg.

**Etap 1: Dashboard (obowiązkowy)**

Widoki dla tego etapu: [general.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/general.png" \t "_blank).

Główna ramka strony składa się z 3 elementów:

* górnego menu (top bar),
* bocznego menu (sidebar),
* głównej treści.

Twoim zadaniem będzie przygotowanie (na podstawie widoku dostarczonego przez grafika) pociętego i responsywnego panelu administracyjnego na podstawie poniższej specyfikacji:

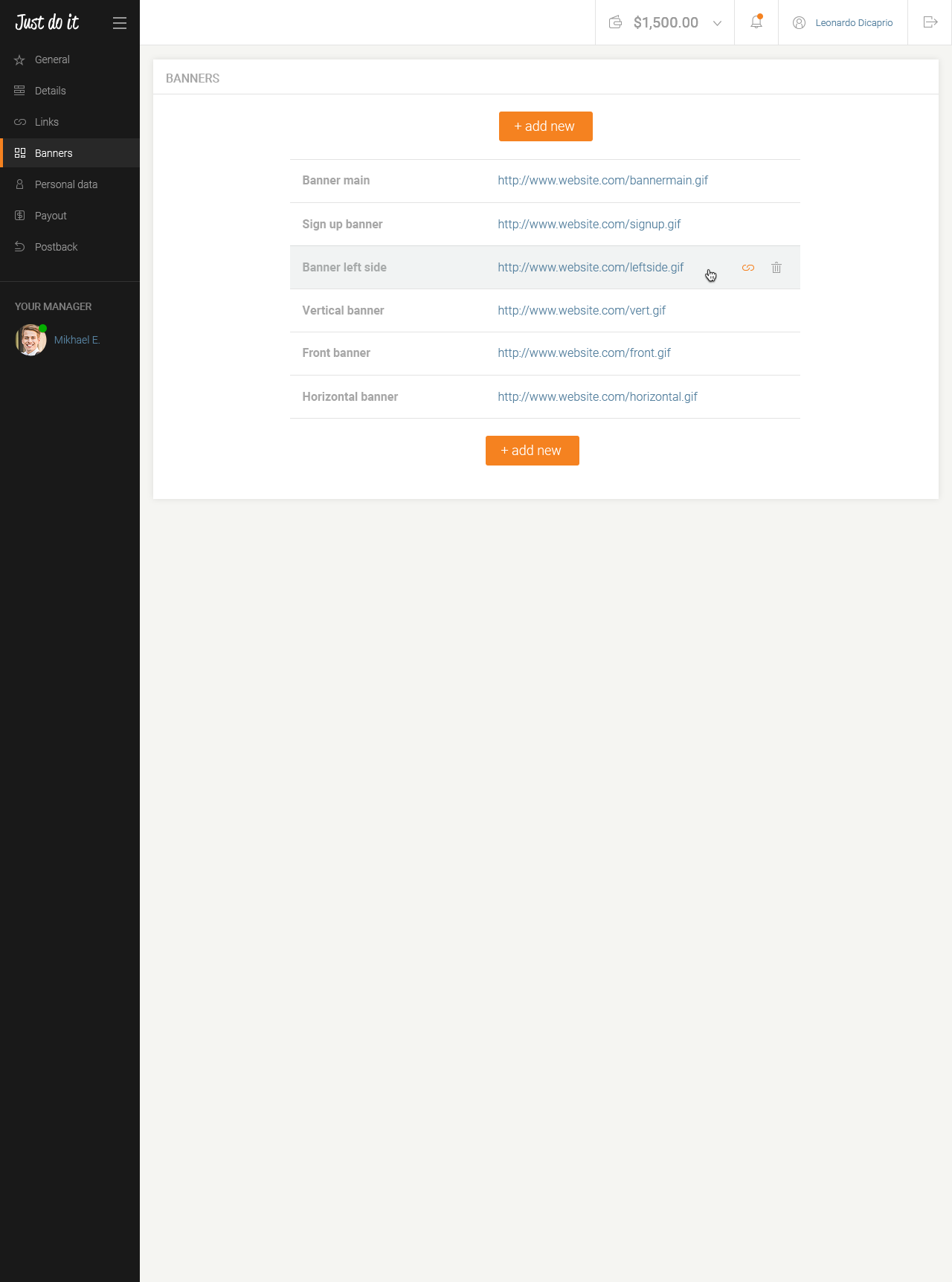
* cały projekt powinien być oparty o samodzielnie stworzony grid,
* wszystkie wykresy powinny być zrobione obrazkami,
* wraz ze scrollowaniem strony, sidebar powinien cały czas być widoczny (position: fixed) po lewej stronie, a górne menu — na górze okna przeglądarki,
* po najechaniu na wiersz w tabeli, ten powinien się podświetlić,
* stosuj ikony dostarczone z projektem,
* na telefonach domyślnie sidebar ma być ukryty — powinien się pojawiać po kliknięciu w przycisk obok menu,
* na telefonach elementy znajdujące się w top barze po prawej stronie powinny się znaleźć w sidebarze na samym dole,
* pamiętaj, aby wszystkie przyciski i linki miały efekt najechania (hover effect).

Możesz samodzielnie zadecydować o wszystkim, co nie zostało tutaj sprecywane. Przy podejmowaniu decyzji kieruj się dobrem projektu - tzn. podejmuj takie decyzje, aby projekt jak najwięcej na nich zyskał.

**Etap 2: Lista linków i bannerów (dla chętnych)**

Widoki dla tego etapu: [links.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/links.png" \t "_blank), [banners.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/banners.png" \t "_blank).

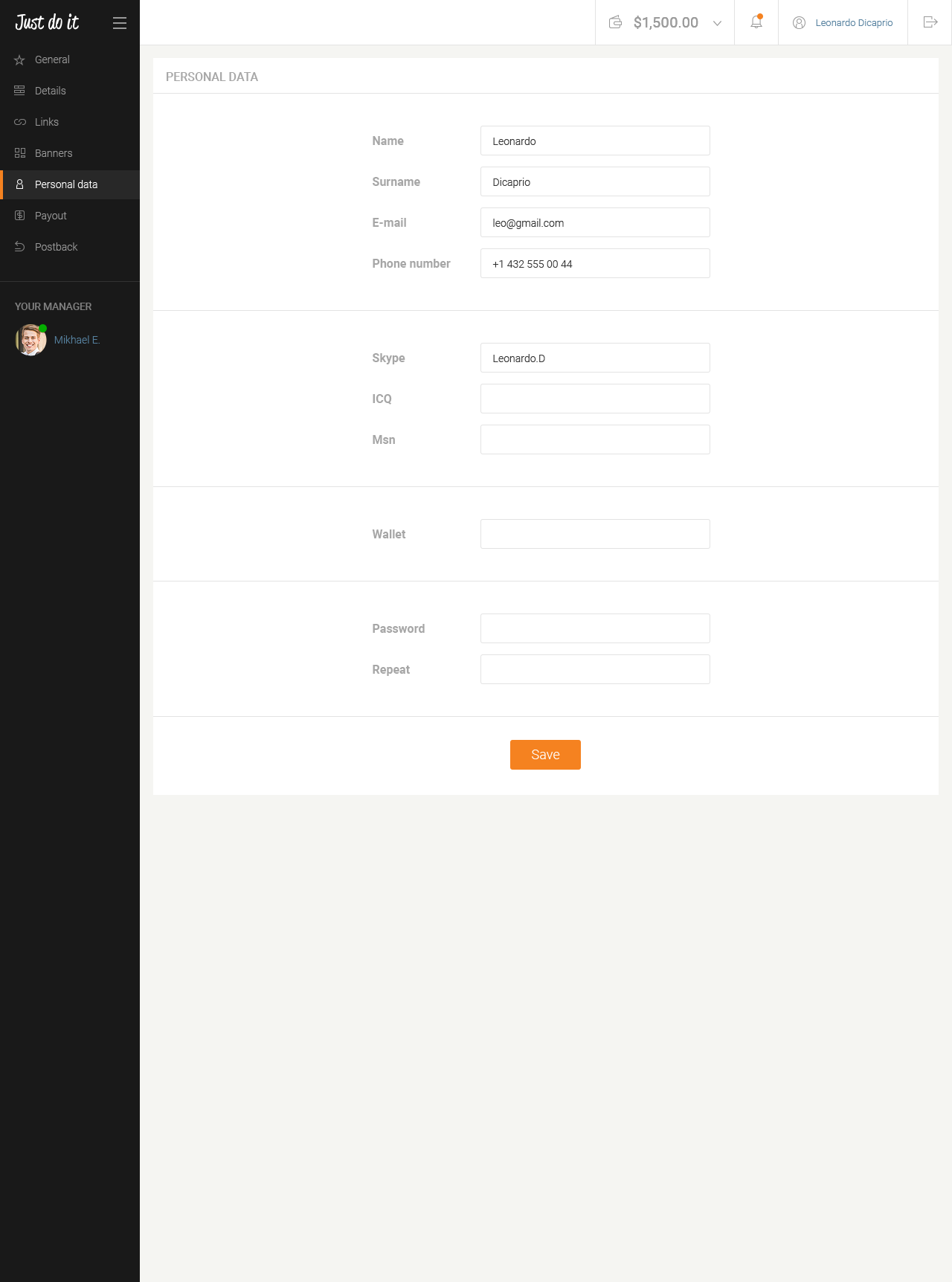
Drugi etap jest prosty, ponieważ polega na przygotowaniu podobnie wyglądających list linków i bannerów. Zauważ, że w poprzednim etapie stworzyliśmy podobnie wyglądające elementy:

[](https://uploads.kodilla.com/bootcamp/wdp/WDP-1-1-2.png)

**Etap 3: Dane osobowe (dla chętnych)**

Widoki dla tego etapu: [personal\_data.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/personal_data.png" \t "_blank).

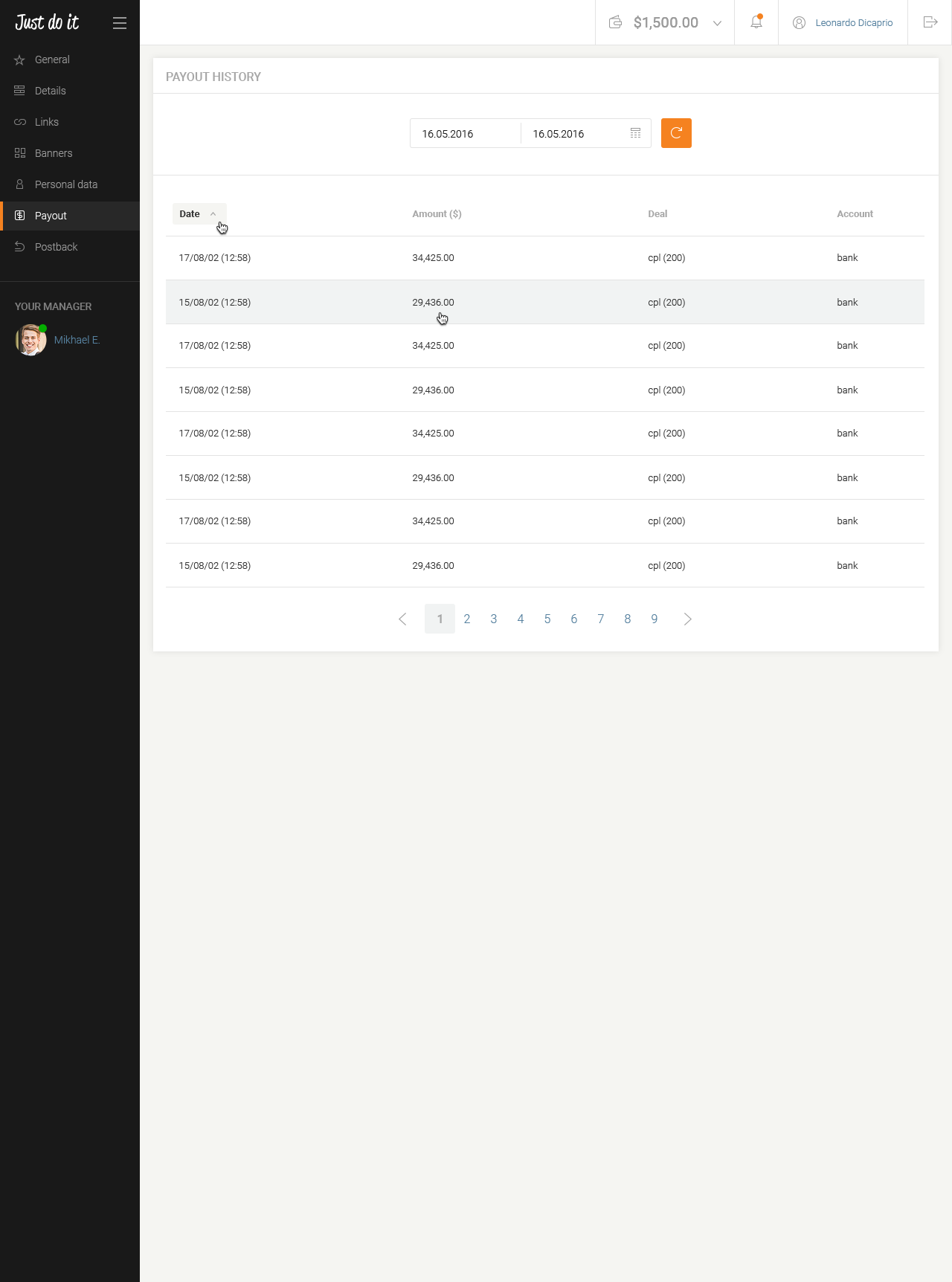
Tym razem będzie trzeba zrobić nieco więcej: Twoim zadaniem będzie przygotowanie formularza danych osobowych. W przypadku urządzeń mobilnych zadbaj, aby label pola znajdował się tuż nad nim.

[](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/personal_data.png)

**Etap 4. Tabele details oraz payout (dla chętnych)**

Widoki dla tego etapu: [details.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/details.png" \t "_blank), [payout.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/payout.png" \t "_blank).

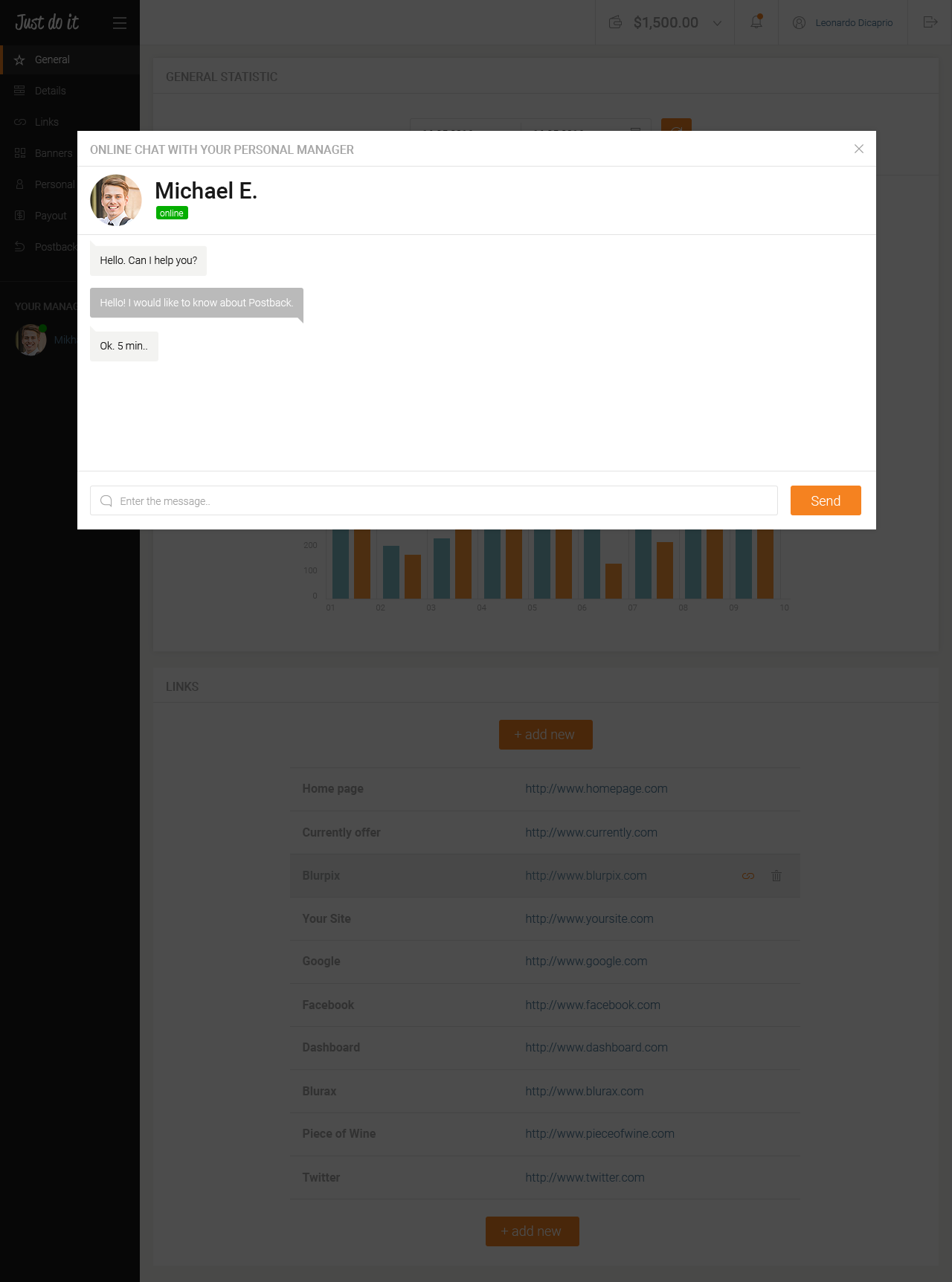
Dodajmy w takim razie kolejne podobnie wyglądające widoki zawierające tabelę z bardziej rozwiniętymi strukturami. Zwróć uwagę, że tabele z dużą liczbą kolumn wymagają innego podejścia przy małych urządzeniach.

[](https://uploads.kodilla.com/bootcamp/wdp/WDP-1-1-5.png)

**Etap 5. Proste pop-upy (dla chętnych)**

Widoki dla tego etapu: [popup\_login.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/popup_login.png" \t "_blank), [popup\_message.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/popup_message.png" \t "_blank), [popup\_quit.png](https://uploads.kodilla.com/bootcamp/wdp/solo-project-1/popup_quit.png" \t "_blank).

Ostatnim etapem naszej pracy będzie stworzenie pop-upów, które uzupełnią nasz projekt. Spróbuj stworzyć ich wywołania po kliknięciu w konkretne linki.

[](https://uploads.kodilla.com/bootcamp/wdp/WDP-1-1-7.png)

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**9.2. Przygotowanie do pracy**

**Repozytorium git**

Na samym początku pracy stworzymy nowe repozytorium git i połączymy je z naszym projektem.

1. Stwórz nowe repozytorium na GitHubie.
2. Sklonuj to repozytorium u siebie na komputerze.

W ten sposób przygotowaliśmy sobie repozytorium do rozpoczęcia prac z plikami.

Jeśli masz problem ze stworzeniem lub korzystaniem z repozytorium, wróć do modułu w którym opisywaliśmy repozytoria zdalne.

**Task runnery**

Kolejną rzeczą, którą musimy sobie przygotować, są task runnery, które umożliwią nam pracę nad projektem. Stworzymy dwie grupy task runnerów: produkcyjne i developerskie. Task runnery produkcyjne będą miały za zadanie przygotować pliki do publikacji: pliki .css będą skompresowane i nie będą zawierać map. Task runnery developerskie będą dbać o poprawność naszego kodu oraz będą kompilować SASS z mapami, tak abyśmy mogli bez problemów rozwijać nasz projekt.

Znajdź plik package.json z jednego ze swoich wcześniejszych zadań i skopiuj do katalogu tego projektu. Następnie wykonaj komendę npm run init-project, aby:

* zainstalować wszystkie potrzebne paczki NPM,
* stworzyć niezbędne pliki i katalogi,
* oraz pobrać .gitignore, który wykluczy pliki wynikowe z repozytorium.

Jeśli nie pamiętasz jak działa nasz task runner, wróć do modułu w którym tworzyliśmy go w oparciu o skrypty NPMa. Najważniejsze to pamiętać, że npm run build generuje nam wersję produkcyjną, a npm run watch - developerską.

**Struktura katalogów i plików**

Zanim przejdziemy do cięcia projektu, będziemy musieli zacząć od stworzenia struktury plików, która umożliwi nam sprawną pracę. Oto proponowana przez nas struktura:

.

├── css

├── images

├── js

│ └── script.js

├── sass

│ ├── components

│ ├── pages

│ ├── sections

│ ├── \_layout.scss

│ ├── \_mixins.scss

│ ├── \_normalize.scss

│ ├── \_variables.scss

│ └── style.scss

├── vendor

├── .gitignore

├── package.json

├── styleguide.html

└── index.html

Teraz opiszemy sobie każdy z plików/katalogów:

* **css** — w tym katalogu będą znajdować się przetworzone przez nas pliki css,
* **images** — w tym katalogu będziemy trzymać wszystkie obrazki użyte w projekcie,
* **js** — tutaj będą znajdować się wszystkie stworzone przez nas pliki javascript,
* **sass** — źródła styli w wersji SASS,
* **sass/components** — katalog z plikami źródłowymi styli każdego komponentu (za chwilę wyjaśnimi czym są komponenty),
* **sass/pages** — katalog z plikami źródłowymi każdego widoku. Każdy z tych plików będzie mówił o specyficznych regułach np. zachowywania się sekcji między sobą,
* **sass/sections** — katalog z plikami źródłowymi styli każdej sekcji,
* **sass/\_layout.scss** — plik z gridem oraz domyślnymi klasami pomocniczymi dla niego,
* **sass/\_mixins.scss** — w tym pliku zbierzemy wszystkie mixiny,
* **sass/\_normalize.scss** — przerobiony na SASS plik [normalize.css](http://necolas.github.io/normalize.css/" \t "_blank),
* **sass/\_variables.scss** — tutaj będziemy definiować wszystkie zmienne dotyczące projektu takie jak kolory, wielkości, czcionki oraz inne standaryzacje,
* **sass/style.scss** — nasz główny plik SASS, w którym będziemy importować wszystkie mniejsze pliki,
* **.gitignore** — tutaj zamieścimy pliki i katalogi, które mają być ignorowane przez gita,
* **vendor** — w tym katalogu będziemy zamieszczać niemodyfikowane pluginy i biblioteki. Mogą tam znajdować się zarówno pliki .css jak i .js,
* **styleguide.html** — księga styli, w której będziemy kodować poszczególne komponenty oraz sekcje,
* **index.html** — główny plik HTML.

Tak przygotowana struktura plików pozwoli nam rozpocząć pracę nad projektem.

**Dodajemy normalize.css**

Normalize.css to mały plik CSS, który zapewnia lepszą spójność między przeglądarkami w domyślnej stylizacji elementów HTML. Jest to nowoczesna, gotowa do użycia w HTML5, alternatywa dla tradycyjnego resetowania CSS.

Cechy normalize.css:

* zachowuje przydatne ustawienia domyślne przeglądarki, zamiast je usuwać,
* normalizuje style dla szerokiej gamy elementów HTML,
* poprawia błędy i typowe niespójności w przeglądarkach,
* poprawia użyteczność dzięki subtelnym ulepszeniom,
* wyjaśnia kod za pomocą komentarzy i szczegółowej dokumentacji,
* obsługuje wiele przeglądarek (w tym przeglądarki mobilne) i zawiera CSS, który normalizuje elementy HTML5, typografię, listy, treści osadzone, formularze i tabele.

Możemy go ściągnąć [stąd](http://necolas.github.io/normalize.css/).

Ale co z tym zrobić? Jak przerobić na SASS? Ręcznie? Nic bardziej mylnego — wystarczy, że zawartość normalize.css wkleimy do \_normalize.scss i na tym kończy się cała praca z tym plikiem.

**Przygotowujemy style.scss**

Ponieważ chcemy mieć tylko jeden finalny plik CSS, to wszystkie pliki SASS powinniśmy połączyć w jeden, a później go przekompilować do CSS. Takim łącznikiem będzie dla nas właśnie plik style.scss.

W pierwszej kolejności przypomnij sobie, w jaki sposób importujemy pliki w SASS. Jak pamiętasz z poprzednich lekcji, wszystkie pliki, poza głównym, mają z przodu podkreślenie(\_), które informuje SASS o tym, że są to pliki, które będą importowane w innych plikach, i które mają być pomijane przy kompilacji jako osobne pliki.

W ten sposób funkcja importowania pliku \_variables.scss wygląda tak:

@import "variables";

Jak widzisz, w kodzie nie używamy ani tego podkreślenia, ani rozszerzenia pliku, aby wskazać jego źródło.

Teraz przyszedł czas na zaimportowanie naszych plików w style.scss

Zaczniemy od zaimportowania plików, które są związane tylko i wyłącznie z SASS, czyli zmiennych oraz mixinów.

@import "variables";

@import "mixins";

**Uwaga!** Kolejność importowania nie jest przypadkowa. Pliki są czytane i interpretowane z góry do dołu — z tego powodu w pierwszej kolejności importujemy \_variables.scss, ponieważ musimy zagwarantować, że każdy z naszych plików będzie mieć swobodny dostęp do zmiennych.

W drugiej kolejności będziemy importować już pliki, które faktycznie pojawią się w naszym CSS. Zaczniemy od \_normalize.scss oraz \_layout.scss:

@import "normalize";

@import "layout";

W dalszej części wraz z pojawianiem się kolejnych komponentów, sekcji oraz stron warto zastosować następującą kolejność

/\* Components \*/

@import "components/form\_field";

/\* Sections \*/

@import "section/registration\_form";

/\* Pages / Views \*/

@import "pages/registration";

**Definiujemy zmienne**

W naszym projekcie chcemy zdefiniować sobie najważniejsze rzeczy takie jak kolory, czcionki itp. Warto w tym celu przygotować sobie zmienne, z których będziemy korzystać w innych plikach.

Jedną z ważniejszych rzeczy przy definiowaniu zmiennych jest stworzenie jasnej, jednoznacznie brzmiącej nazwy, która pozwoli nam na zidentyfikowanie jej w kodzie.

Przykłady zmiennych, które mogą Ci się przydać:

/\* Colors \*/

$paragraph-color: #000000;

$brand-main-color: #ff0000;

/\* Font sizes \*/

$paragraph-font-size: 16px;

/\* Fonts \*/

$brand-font: "Open Sans", Helvetica, Arial;

$paragraph-font: "Open Sans", Helvetica, Arial;

Te wszystkie rzeczy wystarczy wrzucić do \_variables.scss, aby bezproblemowo zacząć ich używać w poszczególnych miejscach.

Jeśli masz wątpliwości do powyższego materiału, to - zanim zatwierdzisz - zapytaj na czacie :)

Zapoznałe(a)m się!

**9.3. Podejście komponentowe**

Mamy już gotowe środowisko i repozytorium. Zatem co teraz? Od czego zacząć i jak się zabrać za właściwe cięcie projektu? Wbrew pozorom, zanim zaczniemy cokolwiek robić, trzeba dokonać wstępnej analizy projektu graficznego. Dopiero później możemy sobie przygotować markup, czyli zarys kodu, który będziemy następnie stylować. Dopiero w ostatnim etapie swojej pracy będziemy łączyć wszystko w całość.

**Analiza projektu graficznego**

Jedną z pierwszych rzeczy rzeczy, jakie powinniśmy zrobić przed rozpoczęciem jakichkolwiek prac, jest analiza projektu graficznego. Polega ona na wstępnym podziale strony na sekcje i komponenty. Zacznijmy od zdefiniowania, czym będzie sekcja, a czym komponent.

**Komponentem** nazwiemy każdy element na stronie, który mógłby być użyty wielokrotnie, a przy tym może zawierać występować w wielu wariantach lub stanach. Przykładem komponentu może być przycisk, pole w formularzu, blok tekstu ze zdjęciem, czy nawet sama nawigacja. Jak pamiętasz z modułu o Bootstrapie, w dokumentacji pokazane były różne sposoby na stworzenie przycisku.

<div class="btn btn-succes">Przycisk</div>

<a href="#" class="btn btn-success">Przycisk</a>

<button class="btn btn-success">Przycisk</button>

Niezależnie od tego, jaką funkcję miał spełniać przycisk, mogliśmy ostylować go w ten sam sposób bez żadnych różnic. Co więcej, mogliśmy tworzyć jego najróżniejsze warianty.

<div class="btn btn-success">Normalny zielony przycisk</div>

<div class="btn btn-danger">Normalny czerwony przycisk</div>

<div class="btn btn-lg btn-success">Duży zielony przycisk</div>

<div class="btn btn-sm btn-danger">Mały czerwony przycisk</div>

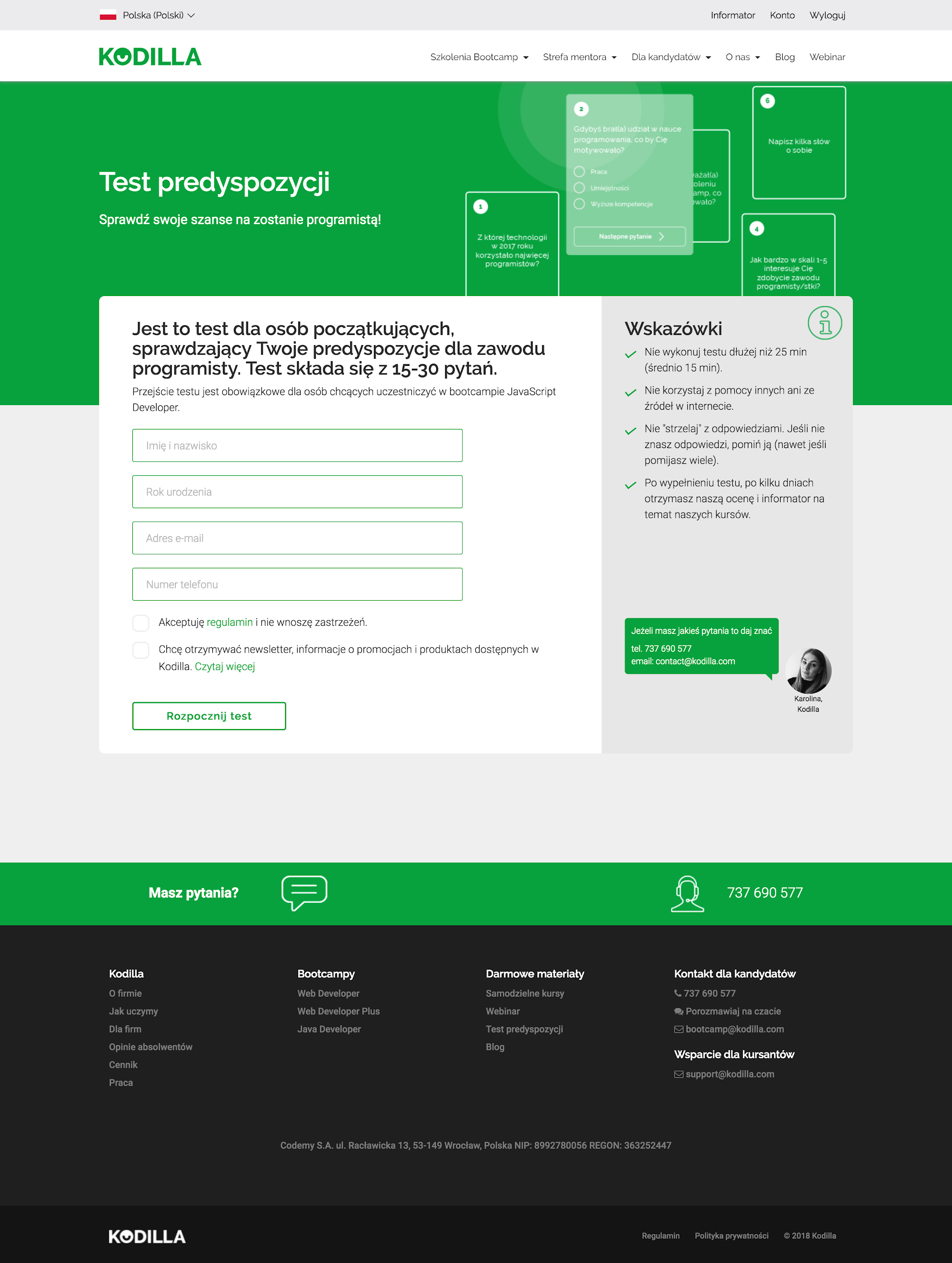
W tym projekcie nie będziemy używać Bootstrapa, ale warto wzorować się na nim w kwestii stosowania komponentów. O ile w Bootstrapie mamy gotowe komponenty, które możemy wykorzystywać, to w naszym projekcie będziemy sami takie komponenty tworzyć.

Dlatego na samym początku warto określić, ile tak naprawdę mamy takich elementów, jakie one są, oraz jakie powinny mieć stany. Wyszukiwanie komponentów sprawi, że będziemy mogli przygotować sobie już na wczesnym etapie tzw. specyfikację/dokumentację projektu. W późniejszych etapach, kiedy mogłyby pojawić się jakieś nowe widoki do przygotowania, będziemy przygotowani na to, żeby móc zbudować stronę z już istniejących komponentów.

Drugim rodzajem elementu będą **sekcje**, które najogólniej możemy określić mianem wrapperów treści. Innymi słowy, **sekcja jest komponentem**, który łączy wiele elementów/komponentów w celu zaprezentowania konkretnej treści. Sekcją na stronie może być formularz rejestracyjny, box do logowania, galeria zdjęć czy lista pracowników firmy. Podział na sekcje jest o tyle istotny, że niestety, czasem graficy planują jakieś nie w pełni spójne z całą resztą układy lub stany komponentów np. nietypowo większy/mniejszy margines między komponentami. Sekcja będzie właśnie takim wrapperem, która umożliwi nam narzucenie tych styli na osadzone w niej komponenty.

**Dzielimy projekt na sekcje i komponenty**

Skoro już wiemy, z jakich elementów może składać się strona, czas przejść do części bardziej szczegółowej. Przyjrzyjmy się naszemu przykładowemu projektowi graficznemu:

[](https://uploads.kodilla.com/bootcamp/wdp/WDP-1-1-9.png)

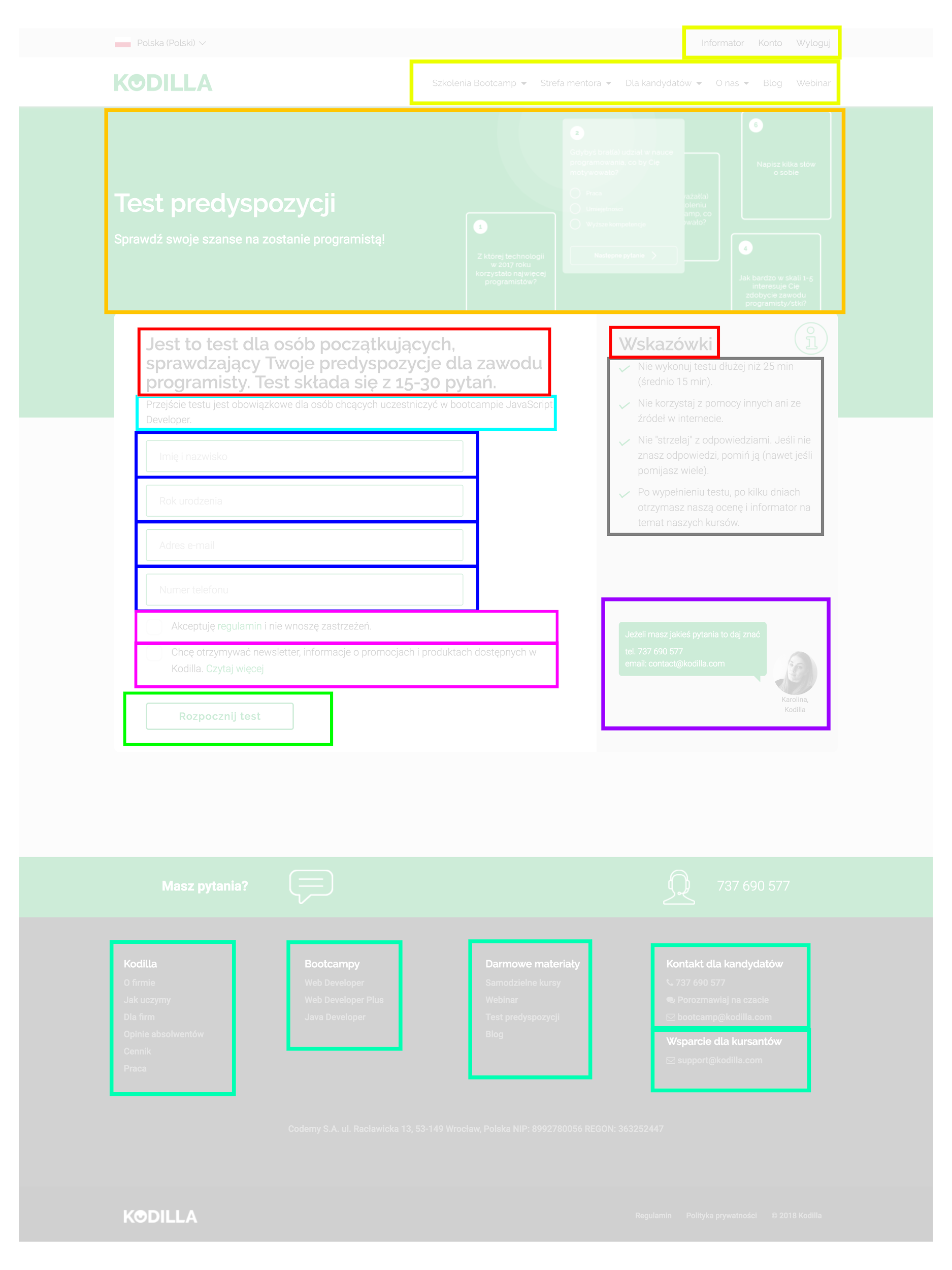
Na tym etapie warto wziąć kartkę papieru i długopis, a następnie rozrysować sobie mniej wiecej układ strony oraz określić, czy coś powinno być komponentem, czy też nie. Jakie kryterium przyjąć do tego celu? Warto zadać sobie następujące pytania:

1. Czy ten element pełni jakąś funkcję na stronie lub ma jakieś zachowania (rozwijane menu)?
2. Czy ten element pojawia się kilkukrotnie na tej stronie?
3. Czy ten element jest używany na tej podstronie lub może być używany na innych podstronach w postaci wariantów?

Rzecz jasna, jeśli zechcesz uczynić logo osobnym komponentem, to nie ma większego problemu — podane kryteria mają wyłącznie ułatwić Ci podejmowanie decyzji.

**Ważne: na tym etapie dzielimy sobie projekt na najmniejsze możliwe komponenty.**

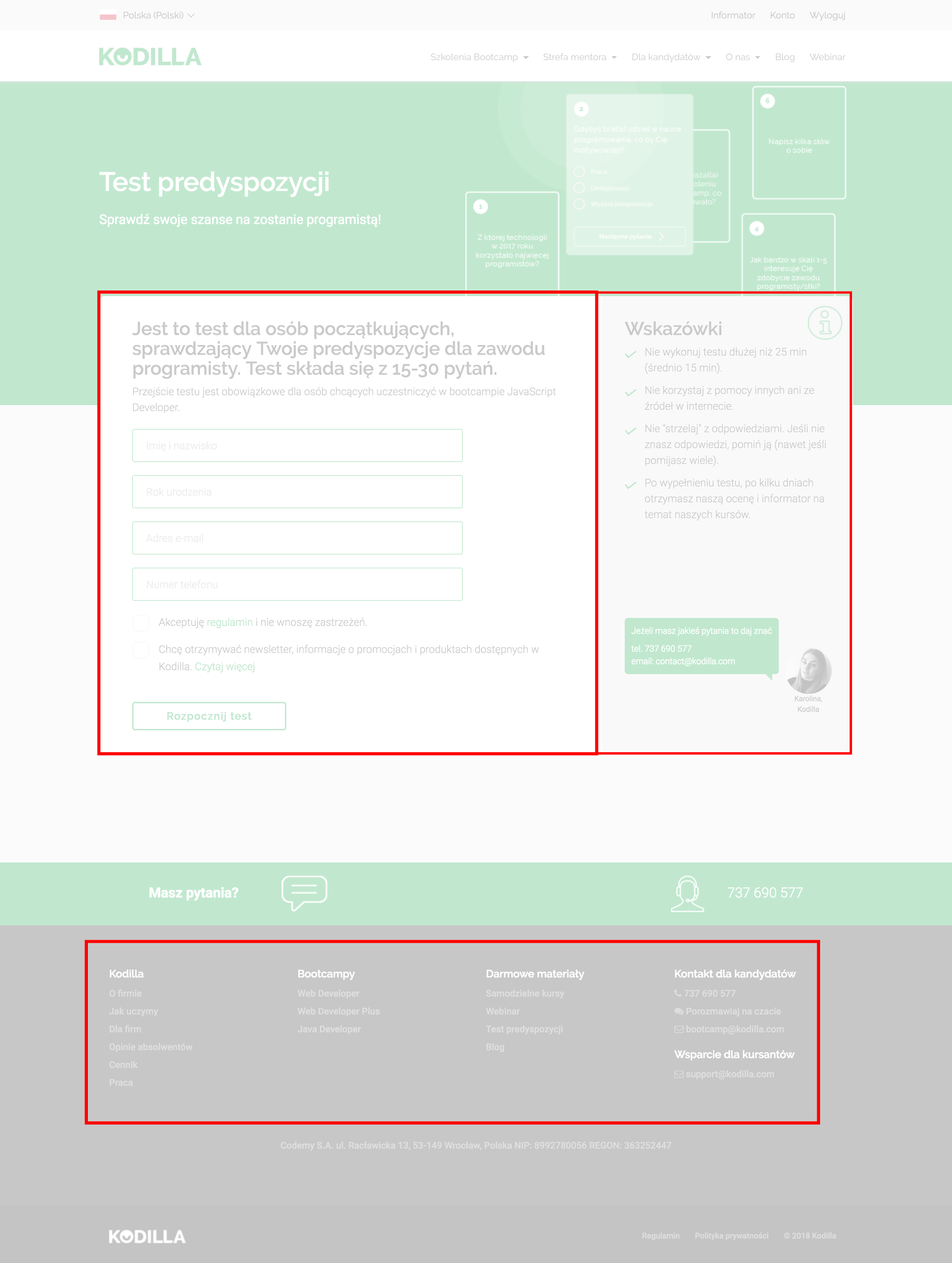
Przykładowy podział może wyglądać następująco:

[](https://uploads.kodilla.com/bootcamp/wdp/WDP-1-1-10.png)

Na tym przykładzie osobnymi kolorami zaznaczyliśmy najmniejsze komponenty. Rzeczy, których nie zaznaczyliśmy, prawdopodobnie pojawiają się **jednokrotnie i są stałą częścią danego komponentu (lub sekcji)** .

Warto dodatkowo zauważyć, że zaznaczyliśmy również nagłówki i paragrafy. Czy będziemy nadawać im np. osobne klasy? Nie, ale będą miały ujednolicone style, stąd już na tym etapie wzięliśmy to pod uwagę.

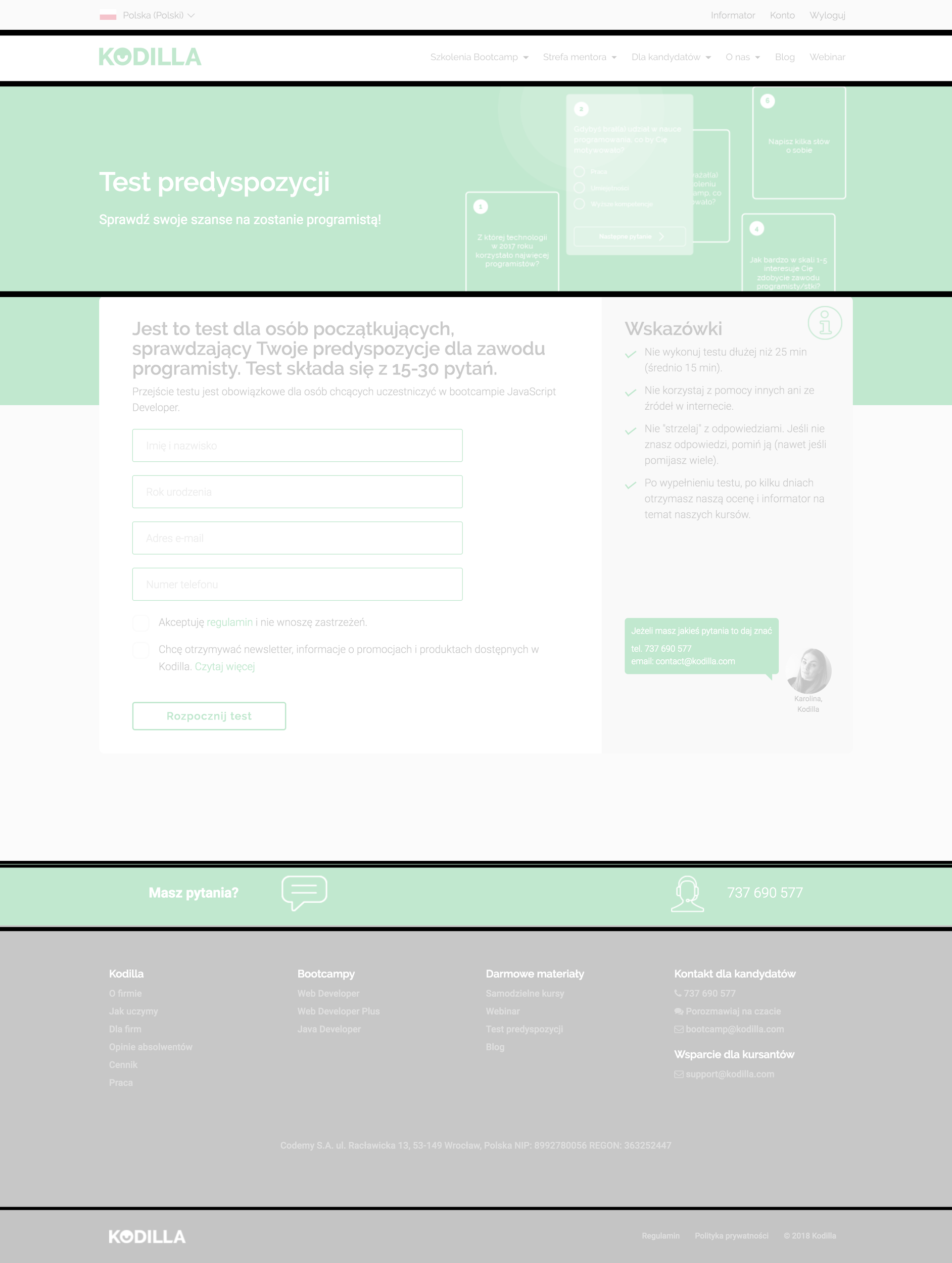
Drugim etapem naszej pracy będzie zbudowanie bardziej złożonych komponentów, takich jak kompletny formularz czy układ kolumnowy. Z takich elementów będziemy później składać całe sekcje.

[](https://uploads.kodilla.com/bootcamp/wdp/WDP-1-1-11.png)

Na koniec będziemy dokonywać podziału podziału na sekcje. Założenie podziału na sekcje jest następujące: czy dana grupa elementów/komponentów łączy je ze względu na wspólną funkcjonalność/treść LUB czy jest jednoznacznie oddzielona wizualnie od innych sekcji.

Najczęściej sekcje na stronach stanową jeden rząd. Przykładowo możemy mieć sekcję z członkami zespołu, sekcję "O produkcie/firmie", czy sekcję kontaktową. Czasem wystarczy graficzne oddzielenie pomiędzy kolejnymi sekcjami strony, np. zmiana koloru tła. Podział na sekcje jest dość indywidualną kwestią i często musimy dokonać podziału zgodnie z własną intuicją.

W naszym przykładzie można zastanawiać się, dlaczego dokonaliśmy podziału treści strony (pomiędzy menu a stopką) na 2 części — nagłówek oraz treść (formularz). Tutaj założyliśmy sobie, że podobny nagłówek może pojawiać się na wielu podstronach i warto, żeby można było go powielać.

[](https://uploads.kodilla.com/bootcamp/wdp/WDP-1-1-12.png)

Nawigacja jest sekcją, ponieważ zawiera logo i menu, które łączą się w ramach jednej funkcjonalności — nawigowania. Ważne jest to, że sekcje możemy wykorzystywać jak komponenty, tj. na różnych podstronach mogą pojawić się te same sekcje, czasem nawet w różnych wariantach.

**Przygotowujemy styleguide**

Im większy projekt, nad którym pracujemy, tym bardziej powinniśmy się postarać o przygotowanie spójnych styli. W tym celu często przygotowujemy **styleguide**, czyli księgę styli, która pozwoli nam zorientować się, jak domyślnie powinny wyglądać najprostsze elementy HTML w różnych przypadkach użycia.

W katalogu src utwórz plik styleguide.html. Zbuduj typową strukturę HTML oraz podepnij style. Wewnątrz tagu <body> umieść następujące elementy:

<h1>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</h1>

<h2>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</h2>

<h3>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</h3>

<h4>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</h4>

<h5>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</h5>

<h6>Lorem ipsum dolor sit amet, consectetur adipiscing elit.</h6>

<p>Lorem ipsum dolor sit amet, consectetur <strong>adipiscing elit</strong>. Maecenas blandit diam urna, eu <a href="#">dignissim</a> ex dignissim consequat. Duis porta lectus lorem, quis eleifend justo scelerisque eget. Duis ac sem et turpis tincidunt molestie in non ipsum. Maecenas ut augue erat. Phasellus eu est commodo est hendrerit tempor. Mauris dignissim ex vel magna aliquet, sit amet facilisis turpis tincidunt. Fusce quis blandit libero, sed dapibus nisl. Pellentesque vitae ligula sed tellus sagittis dapibus eu non metus. Mauris rutrum ligula ut sapien varius, pellentesque volutpat diam porta. Ut at nunc ac tortor dapibus sodales id at leo. Nunc posuere lorem non feugiat elementum. Nunc sagittis diam justo, nec malesuada sapien tristique id. Vivamus in consequat nibh. Etiam tempor id ligula a dapibus. Proin vitae arcu nulla.</p>

<a href="#">Lorem ipsum dolor sit amet, consectetur adipiscing elit.</a>

<ul>

<li>Lorem ipsum dolor, consectetur <a href="#">consectetur adipiscing elit.</a>.</li>

<li>Lorem ipsum dolor <strong>sit amet</strong></li>

<li>Lorem ipsum dolor <strong>sit amet</strong>, consectetur <a href="#">consectetur adipiscing elit.</a>.</li>

</ul>

<ol>

<li>Lorem ipsum dolor, consectetur <a href="#">consectetur adipiscing elit.</a>.</li>

<li>Lorem ipsum dolor <strong>sit amet</strong></li>

<li>Lorem ipsum dolor <strong>sit amet</strong>, consectetur <a href="#">consectetur adipiscing elit.</a>.</li>

</ol>

Mając taką listę podstawowych elementów, możesz zacząć przygotowywać plik z podstawowymi stylami dla strony. Warto tutaj skupić się na wielkości czcionek, typie, grubości, kolorze oraz marginesach.

Jak już wspomnieliśmy wcześniej, dobrą praktyką jest dodanie do projektu pliku [normalize.css](https://necolas.github.io/normalize.css/" \t "_blank). Tego typu paczka sprawia, że przeglądarki wyświetlają wszystkie elementy bardziej konsekwentnie i zgodnie z nowoczesnymi standardami. Dokładnie dotyczy tylko stylów, które wymagają normalizacji.

Pamiętajmy, że te style ustalamy przede wszystkim dlatego, żeby mieć ostylowane tzw. *naked elements*, czyli wolne od dodatkowych klas elementy.

**Przygotowujemy wstępny markup komponentów**

Kolejnym krokiem jest zakodowanie każdego komponentu — możemy to dalej robić **w styleguide.html**. Dzięki temu na jednej stronie będziemy mieli całą bibliotekę wszystkich używanych przez nas komponentów. Swobodnie używaj tutaj dowolnych gridów i/lub br w celu przedstawienia komponentów naszego projektu.

Warto przygotować komponenty z myślą o **typowym przypadku użycia**. Co to znaczy? Jeśli będziemy stylować treść artykułu, to powinniśmy ją testować w ramach grida na dużej liczbie kolumn, lecz jeśli przygotowujemy zawartość sidebara, to lepiej pomyśleć o tym, by ograniczyć nasz testowy grid do 3-4 kolumn.

<div class="container">

<div class="row">

<div class="col-md-4">

<div class="form-field">

…

</div>

</div>

</div>

</div>

Dla ułatwienia, żeby rozróżnić, czy coś jest zwykłym elementem, czy też komponentem, warto użyć tzw. **prefiksów** i zamiast klasy form-field stworzyć klasę component--form-field. Oczywiście wszystko zależy od konwencji, jaką sobie przyjmiemy. Ważne jest jedynie to, by raz przyjętej konwencji nie zmieniać w trakcie kodowania — chyba że liczymy się z potrzebą dostosowania do niej wszystkich innych elementów.

Już na tym etapie warto też pomyśleć o zachowaniach naszych komponentów dla różnych szerokości. Dlatego komponenty przygotowujemy w jednym z następujących podejść:

1. komponent ma zajmować całą dostępną szerokość,
2. komponent ma mieć stałą szerokość minimalną ALBO maksymalną,
3. szerokość komponentu ma zależeć od jego treści (np. wielkość buttona zależy do treści w nim zawartej).

Zwróć także uwagę, aby komponent nie *rozsypywał się*, kiedy zostanie do niego wpisana treść dłuższa niż w projekcie. W takiej sytuacji np. tekst na guziku może być zawijany do dwóch linijek, a w ostateczności obcinany (overflow: hidden).

Tutaj możesz postawić pytanie: czy komponent powinien znajdować się wewnątrz kolumny grida, czy może sam powinien być jego kolumną? Początkującym odpowiemy: nie mieszaj. Możesz użyć grida jako zawartości komponentu, albo komponent wstawić do grida, ale **komponent nie może być sam w sobie kolumną**.

Ale czy na pewno nie może? No oczywiście, że może, ale tematyce osadzania komponentu do grida można by poświęcić cały moduł. Zrobienie komponentu jako kolumny wymaga zrozumienia, kiedy możemy tak zrobić, a kiedy nie, dlatego mniejszym błędem będzie zawsze robienie ich osobno.

**Ważne: stylując komponenty nie uzależniaj styli od klas grida. Wynika to z faktu, że może się zdarzyć sytuacja, że zmieni się wielkość lub układ kolumn, a Twoje style będą wymagać dodatkowych poprawek.**

**Rozszerzamy nasze komponenty**

Drugim etapem pracy nad każdym komponentem będzie stworzenie jego wariantów. W tym samym pliku przygotujmy sobie osobne wywołania tego samego komponentu dla różnych wariantów:

<div class="btn btn-success">Normalny zielony przycisk</div>

<div class"=btn btn-danger">Normalny czerwony przycisk</div>

<div class="btn btn-lg btn-success">Duży zielony przycisk</div>

<div class="btn btn-sm btn-danger">Mały czerwony przycisk</div>

Ciekawym rozwiązaniem może być tutaj użycie konwencji BEM — Block Element Modifier. Jest to konwencja, która mówi, że mamy blok, który zawiera wiele elementów, którego warianty można opisywać za pomocą klas modyfikujących. Jest to dość interesująca konwencja często wykorzystywana w skalowalnych projektach. Jeśli zdecydujesz się na korzystanie z konwencji BEM, opieraj się o jej [dokumentację](http://getbem.com/naming/).

**Łączymy wszystko w sekcje**

Kiedy mamy już opracowane poszczególne komponenty, czas połączyć je w większe sekcje. Tutaj możemy potraktować sekcje podobnie jak komponenty, lub wydzielić je do osobnego folderu i każdej sekcji poświęcić osobny plik. W ramach sekcji warto zadać o pełną ich responsywność. Pamiętajmy, że sekcja nie składa się wyłącznie z komponentów, ale również ze zwykłych elementów. Oczywiście dla rozróżnienia sekcji od komponentu możemy zastosować prefix, np. section--.

**Budujemy finalny layout**

Kiedy cały nasz projekt jest już podzielony na sekcje i komponenty, style się wszędzie zgadzają, a sekcje są w 100% responsywne, możemy połączyć gotowe sekcje w jeden plik.

W przypadku tego projektu warto stworzyć ogólny layout z zafiksowanymi (position: fixed) elementami sidebara oraz górnego menu. Oprócz nich będzie jeszcze potrzebny kontener na treść, w który będziemy wstawiać nasze sekcje.

**Dobra praktyka:** warto do atrybutu body dodać klasę określającą dany layout, np.

<body class="registration-form-template">

…

</body>

W ten sposób możemy sobie stworzyć kolejny plik .scss, który obsłuży nam specyficzne zachowania na danej templatce.

**Uwagi końcowe**

Rzeczą jasną jest, że przed rozpoczęciem prac nad stylowaniem komponentów warto mieć już przygotowane w pełni funkcjonalny grid. Następnie przed rozpoczęciem cięcia każdej sekcji warto zastanowić się, jaki ma ona układ — czy jest to układ typu boxed czy fluid. Pomoże nam to później lepiej używać grida.