Nazwa projektu - opis architektury

|  |  |  |  |
| --- | --- | --- | --- |
| **Wersja** | **Data utworzenia** | **Data ost. modyfikacji** | **Autorzy** |
| 1.0 | dd.mm.rrr | dd.mm.rrrr | Lista autorów |

1. Cel dokumentu

*[Wstępny opis wprowadzający w założenie dokumentu, np.:]*

Celem dokumentu jest przedstawienie wstępnej koncepcji architektonicznej systemu Nazwa projektu. Niniejszy opis należy traktować jako propozycję różnych rozwiązań oraz analizę ich zalet oraz wad.

# 2. Ogólne założenia

*[Opis głównych założeń architektonicznych wynikający z charakterystyki projektu, wymagań (szczególnie pozafunkcjonalnych) oraz ustaleń z klientem. Przykładowo: typ aplikacji, sposób komunikowania się, sposób identyfikowania użytkowników itd. Przykład poniżej.]*

Na podstawie rozmów z klientem i dostarczonych materiałów, przy przygotowaniu architektury kierowano się następującymi założeniami:

* klient posiada serwis (zwany dalej “serwisem X”) wykonany w PHP
* system ma być aplikacją mobilną dostępną na systemach Android i iOS i w sklepach Google Play oraz App Store
* aplikacja mobilna ma komunikować się z pewnymi systemami, w tym z serwisem X
* system musi być dostępny w wielu wersjach językowych (początkowo w polskiej i angielskiej)

# 3. Rejestr decyzji architektonicznych (ADR)

*[Podczas wstępnego projektowania architektury (na bardzo wysokim poziomie), podejmowane są główne decyzje, jak np. zastosowanie MVC, protokołu API, użycie konkretnego języka, frameworka, systemu bazy danych itd. Niektóre z tych wyborów są oczywiste, ale są też takie, które po czasie mogą budzić wątpliwości. W dodatku klient ma prawo poprosić o uzasadnienie decyzji. Stąd przydatne może być prowadzenie rejestru decyzji architektonicznych (ang. Architectural Decision Records), w którym dokumentuje się decyzje, najczęściej w formie tabelarycznej (każda decyzja osobno). Możliwe jest także rozszerzanie rejestru o bardziej szczegółowe pozycje, już w trakcie implementacji.]*

Poniżej przedstawiono opis propozycji decyzji architektonicznych w oparciu o powyższe założenia i doświadczenie zespołu IT.

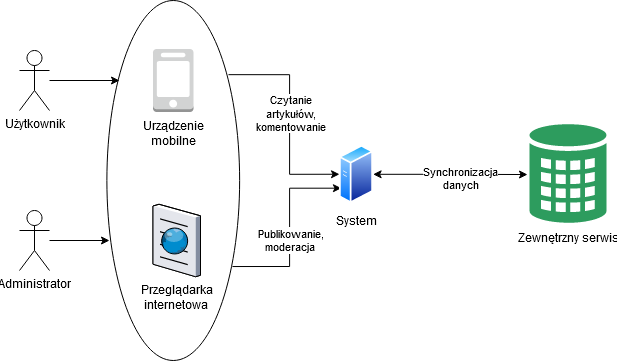
| **Identyfikator** | ADRXYZ |
| --- | --- |
| **Decyzja** | Zastosowanie XYZ |
| **Część architektury** | Frontend/backend/aplikacja/baza danych/style... |
| **Uzasadnienie** | … (uzasadnienie swojego wyboru) |
| **Czynniki ryzyka** | … (możliwe efekty uboczne i konsekwencje) |
| **Inne opcje i konsekwencje** | … (inne rozważane opcje wraz z ich cechami) |

# 4. Diagram kontekstu

*[Jest to pierwszy z diagramów, opisujący architekturę zgodnie ze zmodyfikowaną metodyką C4 Model, w której kolejne “C” dotyczą kolejnych perspektyw: Context, Containers, Components, Code. Ostatnia, ze względu na swoją szczegółowość, nie jest wykorzystywana w tym szablonie, szczególnie, że ten dokument powstaje przed rozpoczęciem prac nad projektem.*

*Diagram kontekstu prezentujące układ i połączenie kontenerów na systemowym poziomie, tzn. relacje pomiędzy poszczególnymi częściami (np. aplikacje klienckie, serwerowe, bazy danych), fragmenty znajdujące się na serwerze, integracje z zewnętrznymi aplikacjami itd. Jest to ogólne pokazanie liczby kontenerów, a więc także rozmiar projektu, w formie graficznej.]*

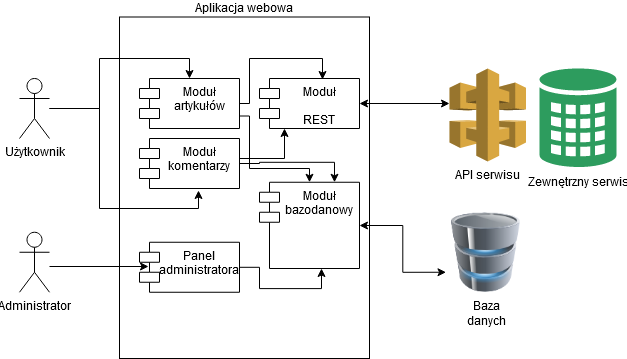
Rozpatrując środowisko biznesowe i kontekst systemu Nazwa Projektu, można przedstawić go na poniższym diagramie kontekstu.

**

# 5. Diagram kontenerów

*[Drugi z diagramów modelu C4. Służy on do dokładniejszego pokazania poszczególnych kontenerów, sugerując np. ich wewnętrzny podział na moduły. W przypadku bardzo rozbudowanych architektur dopuszczalne jest utworzenie kilku diagramów, po jednym na każdy kontener. Uwaga - “kontenera” nie należy tutaj rozumieć jako jednostki w znaczeniu konteneryzacji (np. dockerowej), choć w praktyce może mieć to ze sobą całkiem dużo wspólnego.]*

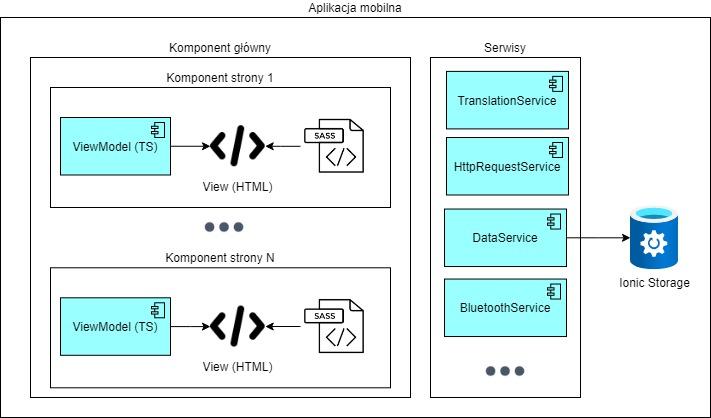
Uszczegóławiając wizję techniczną, system można podzielić na następujące kontenery.



# 6. Diagram komponentów

*[Trzeci z diagramów modelu C4. Prezentuje strukturę wewnętrzną komponentów, jeśli jest to możliwe. Najczęściej znajduje się tu kilka diagramów. Diagram jest użyteczny głównie w momencie, jeśli chcemy pokazać wewnętrzną strukturę np. frontendu, a tym samym obsługującego go frameworku. Nie zawsze ten diagram jest wymagany, szczególnie gdy jego przygotowanie zajmuje zbyt dużo czasu.]*

Decydując się na aplikację mobilną tworzoną przy użyciu frameworka Angular, odpowiednia struktura komponentów oprogramowania przedstawia się następująco:



# 7. Zarządzanie wersjami oprogramowania

*[Opisanie sposobu przechowywania kodu i innych zasobów związanych z oprogramowaniem. Dla klienta jest to informacja świadczą o profesjonalizmie zespołu IT oraz umiejscowieniu kodu, natomiast dla software house’u - wskazówka dotycząca nazywania gałęzi. Przykład:]*

Oprogramowanie będzie powstawało przy pomocy systemu kontroli wersji Git w repozytorium znajdującym się na serwerze zespołu IT. Proponuje się następujący podział i zależność gałęzi:

* master - zawiera aktualną wersję produkcyjną, przekazaną do opublikowania
* beta - zawiera aktualną wersję testową, przekazaną klientowi do testów
* develop - zawiera scaloną powstającą wersję kodu jako zbiór zmian różnych programistów
* pozostałe gałęzie - wersje kodu poszczególnych programistów lub wersje eksperymentalne

# 8. Uwagi końcowe

*[Ewentualne uwagi, np. zachowanie w przypadku wystąpienia jakiegoś przypadku opcjonalnego. Często ten rozdział jest pusty.]*

Brak uwag końcowych.