Management projektů

Návrh systému

Autor: **Pavel Šesták** (**xsesta07**)

24. března 2023

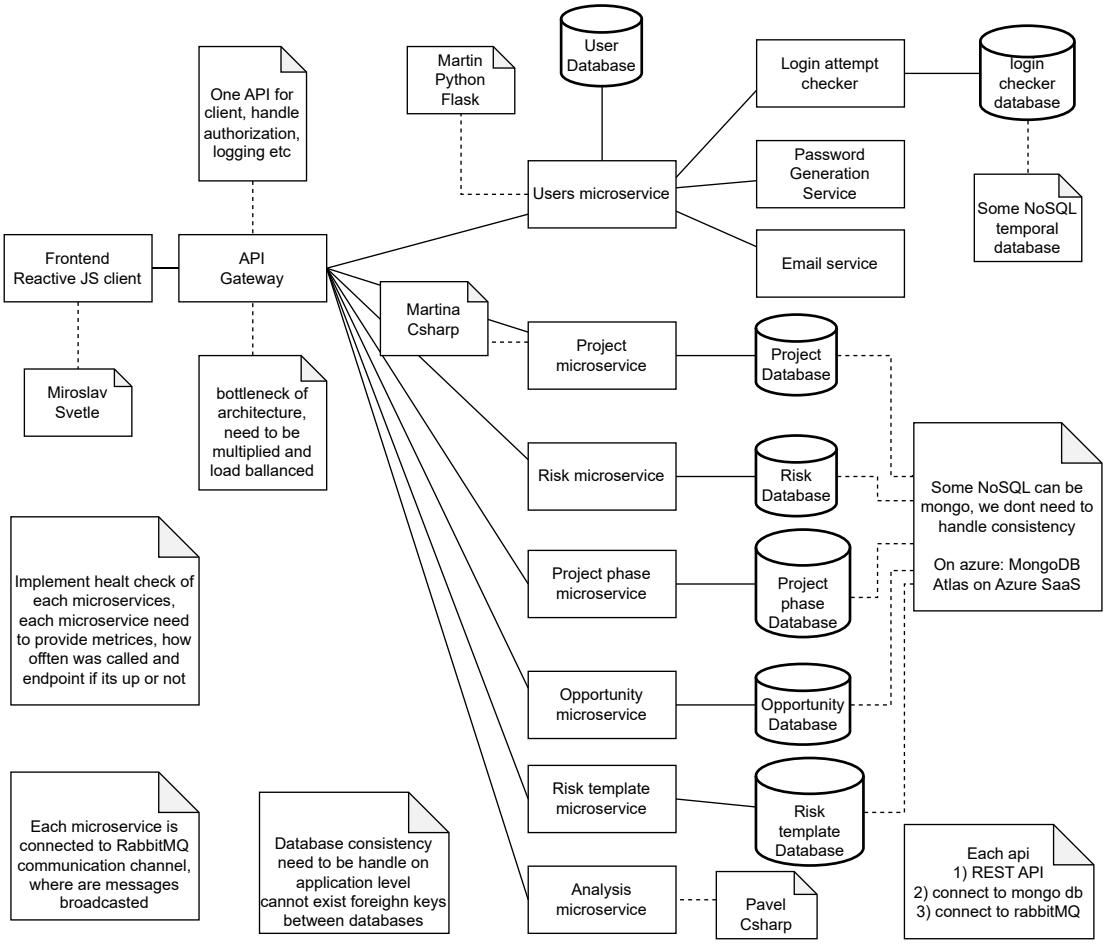
Obsah

1 Úvod

Cílem tohohle dokumentu je shrnutí a popis návrhu systému, který vznikl z dokumentu popisující formální požadavky systému. Návrh je modelován grafickým jazykem UML a využívá jeho standardní notaci. V dokumentu popisující formální požadavky jsme se soustředili na modelování interakcí. V tomto dokumentu se podíváme primárně na strukturu systému.

2 Architektura systému

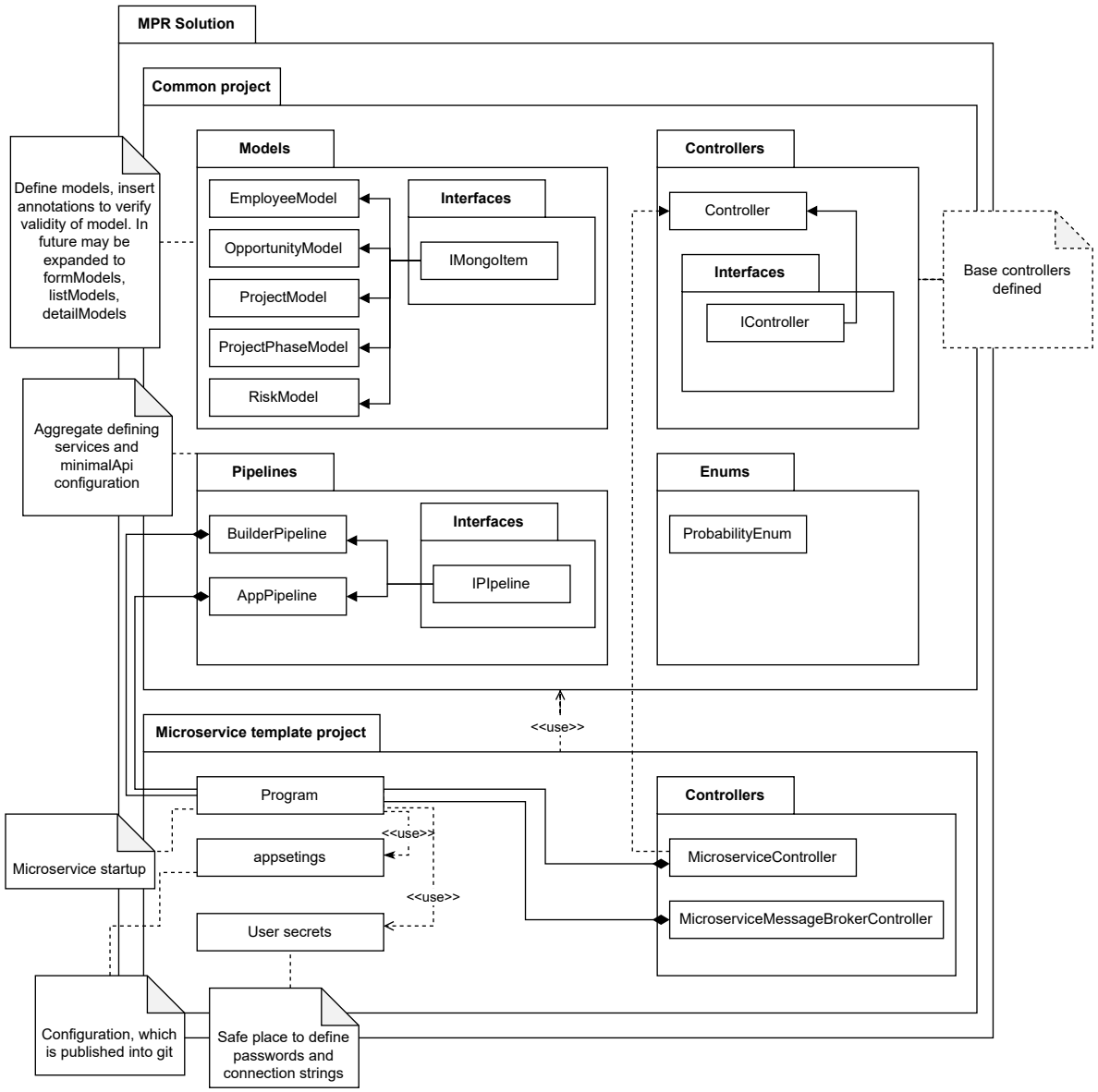
Architektura systému je navržena pomocí mikroslužeb. Problém se povedlo rozdělit na několik separátních mikroslužeb, kde každá entita v systému má vlastní databázi. Přístup k jedné databázi z více služeb se nedoporučuje v případě použití mikroslužeb. Z důvodu rozdělení entit do více databází nelze využít definici vztahů pomocí cizích klíčů na databázové úrovni. Konzistenci dat bude nutné řešit na aplikační úrovni. Jelikož nelze využít relace mezi entitami, tak je zbytečné využívat relační databáze a můžeme sáhnout po rychlejších NoSQL databázích. Z principu distribuce logiky mezi mikroslužby bude nutné zajistit komunikaci mezi jednotlivými službami. Například pokud projektový manažer bude chtít seznam svých projektů, tak kontaktuje *Project microservice*, která bude muset kontaktovat *User microservice*, aby uživatel mohl být autorizován. Další příklad naznačuje problému s konzistencí dat. Projektový manažer maže projekt z databáze, ale k tomu projektu jsou navázány rizika, fáze a příležitosti, které je nutné také smazat, aby nám nezůstali v databázi. Komunikace bude zajištěna pomocí RabbitMQ, což je message broker. Mikroslužby mohou na jednotlivé kanály posílat zprávy s daty a také mohou registrovat odběr na nějakou zprávu ze specifikovaného kanálu.



Obrázek 1: Diagram architektury systému

3 Architektura mikroslužeb

V této sekci si představíme architekturu jednotlivých mikroslužeb. Z důvodu, že většina mirkoslužeb je plánována na platformě dotnet, tak tento diagram popisuje architekturu právě na této platformě, nicméně pro ostatní platformy se dá předpokládat podobné řešení. Všechny mikroslužby budou definovány v rámci jednoho řešení. Visual studio je nakonfigurováno, že v případě debugu automaticky nasadí a spustí mikroslužby do lokálního dockeru. Každá mikroslužba bude samostatný projekt v rámci řešení a bude mít separátní projekt pro testování dané mikroslužby (není modelováno v diagramu). Pro testování mikroslužeb bude využit framework xUnit případně nUnit na základě preferencí vývojářů. Společné definice a typy pro všechny mikroslužby jsou agregovány do projektu Common. Součástí studie proveditelnost byla implementace dvou mikroslužeb, které zapisovali do NoSQL databáze a komunikovali spolu pomocí RabbitMQ. Tyto projekty slouží jako referenční projekty pro programátory.



Obrázek 2: Diagram architektury mikroslužeb dotnet

3.1 Popis architektury projektu Common

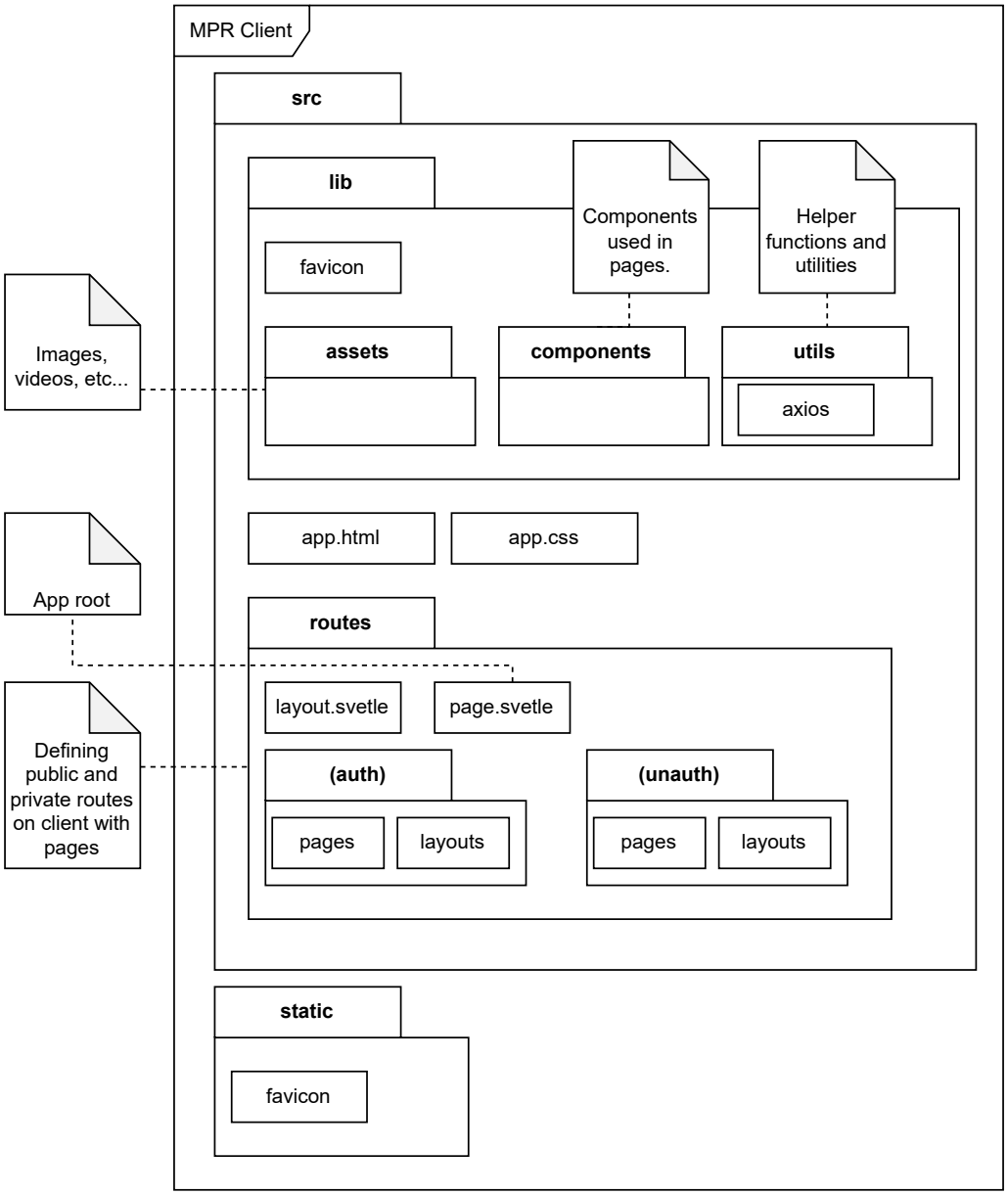
Projekt common se skládá ze čtyř hlavních modulů. Modul Models obsahuje definici datových objektů, které se vyskytují v rámci systému. Podle potřeby bude rozděleno na různé modely pro různé pohledy systému, jelikož pro tabulku nad danou entitou můžeme potřebovat jiné datové položky než například pro formulář editace. Modul kontrolerů obsahuje definici rozhraní a bázový kontroler, jelikož se dá předpokládat, že jádro problému bude dost podobné ať se jedná o projekt nebo jeho riziko. Modul Pipelines obsahuje vytknutou definici registraci služeb pro mikroslužbu a konfiguraci aplikace. Modul enums slučuje použité výčtové typy.

3.2 Popis architektury mikroslužby dotnet

Srdcem každé mikroslužby, která je implementovaná jako dotnet minimal api je soubor Program. V tomto souboru probíhá startování aplikace a pro definici služeb a správnou konfiguraci používá pipeliny z projektu Common. Mapování HTTP endpointů RESTového rozhraní na logiku aplikace je definováno v MicroserviceController. MicroserviceMessageBrokerController slouží pro přihlášení k odběru zpráv z message brokera a implementci adekvátních obslužných rutin.

4 Architektura klientské části

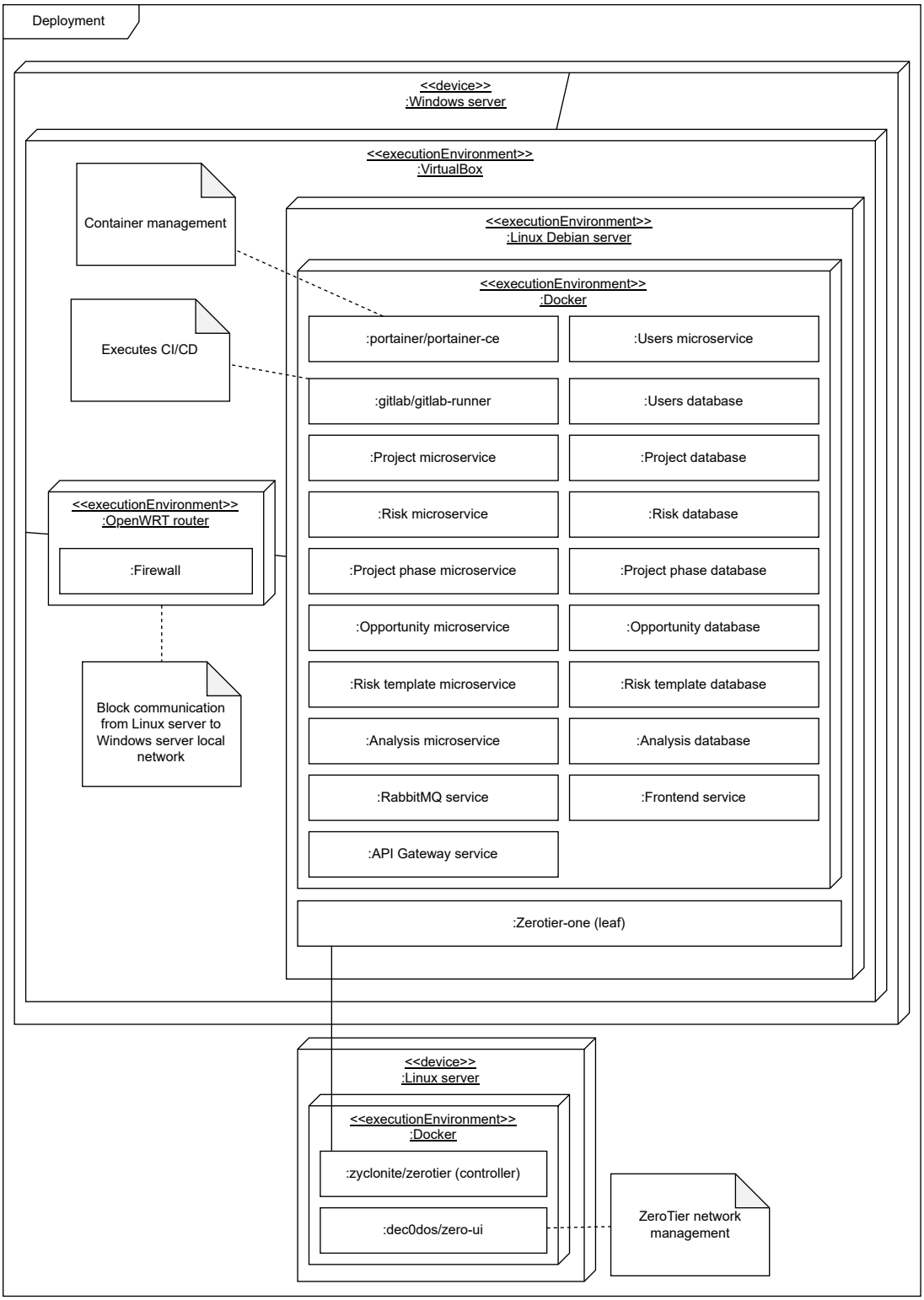
Pro implementaci klientské části byl zvolen reaktivní JS framework Svetle z důvodu preference vývojářů. Architektura je rozdělena na dvě hlavní části kde v rámci modulu lib jsou definovány pomocné komponenty pro jednotlivé pohledy a statické soubory. Modul routes definuje endpointy klientské aplikace, kontroluje autorizaci uživatele a definuje jednotlivé systémové pohledy.



Obrázek 3: Diagram architektury klientské aplikace

5 Nasazení aplikace

V této sekci se zaměříme na nasazení a provoz systému. Systém bude nasazen na Windows serveru ve virtualizovaném prostředí. Ve virtuálním prostředí běží dva servery. Aplikace bude nasazena na linuxové distribuci Debian, který komunikuje skrz openWRT firewall, který je virtualizovaný na stejném windows serveru. V Debianu je nasazen docker, kde jsou nasazeny jednotlivé služby aplikace. Spolu s kontajnery jednotlivých služeb zde běží i kontajner pro správu jednotlivých kontajnerů portainer, a zerotier-one, který slouží pro připojení do sítě, kde bude aplikace dostupná, protože server nedisponuje veřejnou IP adresou. Jelikož je aplikace vyvíjena jako sada mikroslužeb, tak je vhodné, aby se pravidelně nasazovala nová verze služby bez závislosti na ostatních. K tomuto účelu zde slouží kontajner gitlab-runner, který v případě mergnutí větve do mainu provede build. Jakmile projdou automatické testy, tak upravenou službu automaticky nasadí na server. Díky tomuto může nasazování aplikace být plynulé a může vyjít nová verze i několikrát denně bez toho aniž by si toho uživatelé všimli či byla celá služba nedostupná.



Obrázek 4: Diagram nasazení aplikace

6 Závěr

Postupně jsme si představili architekturu celého systému a do detailu rozebrali její části ze strukturálního pohledu. Modelovat popis interakce se systémem není cílem tohoto dokumentu, jelikož tato část je rozebrána v dokumentu formální specifikace požadavků.