5. Проектиране

5.1 Архитектура

Софтуерната архитектура е съвкупност от важни решения за организацията на програмните системи. Тя се определя от високо ниво структури от софтуерни системи, дисциплината от създаването на такива структури и документация на тези структури. Тези структури са необходими за аргументиране на софтуерните системи. Всяка структура съставлява софтуерни елементи, връзки между тях и свойствата на двете - елементи и връзки. Архитектурата на софтуера е метафора, аналогично е на [архитектура](https://bg.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0) на сграда.

Чрез Софтуерната архитектура се вземат структурни решения, които са скъпи за промяна след реализация. Софтуерната архитектура включва специфични структурни варианти от възможности в дизайна на софтуера. Например, системите които контролират космически совалки които трябва да са много бъзи и много надеждни. Следователно, необходимо е да се избере подходящият компютърен език. В допълнение, за да задоволи нуждите за надеждност би могло да се избере на-добрият вариант, да има множество излишни и независими произведени копия на програмата и да се задействат тези копия, докато тече многократна проверка на резултатите.

Документирането на софтуерната архитектура улеснява комуникацията между заинтересованите страни, позволява ранни решения за проектиране на високо равнище и повторно използване на проектните компоненти.

Характеристики на една софтуерна архитектура са:

* Архитектурата спомага за използваемостта. Тя дава възможност на потребителя да поема инициативи например, да прекрати дълго изпълняващи се операции или да се откаже от последната команда;
* Системата трябва да може да придвижва реакцията на потребителя;
* Системата трябва да дава възможност на потребителя да бъде ефективен в работата си чрез прилагане на тактики;
* Нуждата от софтуерната архитектура е предизвикана от необходимостта за солидна основа при големите и сложни системи. Тя е гаранцията за дълъг живот на системата;
* Ако не са предвидени различните сценарии и начини на използване на системата, общите проблеми и начините на справяне с тях, целите в дългосрочен план, то тогава софтуерният продукт е в риск. А това струва време и пари! Разбиране на системата от всяко заинтересовано лице, нейното структуриране на елементите е друга необходимост;
* Софтуерната архитектура играе ролята на инструмент за комуникация, обосновка при вземане на решения, средство за анализ и развитие на системата;

Софтуерната архитектура е един вид“интелектуално разбираема“ абстракция на сложна система. Тази [абстракция](https://bg.wikipedia.org/wiki/%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%86%D0%B8%D1%8F) осигурява редица предимства :

* Дава основа за анализ на поведението на софтуерната система, преди тя е да е била построена. Способността да се провери, че една бъдеща софтуерна система отговаря на нуждите на своите заинтересовани страни без тя да е построена още, представлява значително намаляване на разходите и рисковете.
* Тя осигурява основа за повторно използване на елементи и решения. Пълна софтуерна архитектура или част от него, може повторно да се използва в няколко системи, чиито участници изискват подобни характеристики и функционалности. Това спестява разходи за проектиране и намалява риска от проектантски грешки.
* Тя подкрепя ранните дизайнерски решения, които влияят на разработката, разгръщането и живота на поддръжката. Вземането на ранни решения предотвратява преразходи в бюджета.
* Улеснява комуникацията на заинтересованите страни, допринася за една по-добра система отговаряща на нуждите им.
* Софтуерната архитектура спомага управлението на [риска](https://bg.wikipedia.org/wiki/%D0%A0%D0%B8%D1%81%D0%BA). Тя помага за намаляване на шансовете за провал. Дава възможност за намаляване на разходите в сложни IT проекти.

## 5.2. Модел на данните

Първата задача пред един дизайнер на бази данни е да произведе концептуален модел на данните, който отразява структурата на информацията, която ще се проведе в базата данни. Един общ подход към това е да се разработи модел на обектни взаимовръзки, често с помощта на инструменти за рисуване. Друг популярен подход е [Unified Modeling Language](https://bg.wikipedia.org/wiki/Unified_Modeling_Language). Един успешен модел на даните акуратно ще отрази възможното състояние на външния свят, който е обект на моделирането: например ако хората могат да имат повече от един телефонен номер моделът ще позволи на тази информация да бъде съхранена.

Моделът на база данни е вид модел на данните, който определя логическата структура на базата данни и фундаментално определя по какъв начин данните да се съхраняват, организират и манипулират.

За Trip Planner ще се използва MongoDB. Това е система за обработване на [бази данни](https://bg.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8) от документи, разработена от [10gen](https://bg.wikipedia.org/w/index.php?title=10gen&action=edit&redlink=1). Тя е от рода на [нерелационните бази данни (NoSQL)](https://bg.wikipedia.org/wiki/NoSQL). Вместо да съхранява информация в таблици, както е при традиционните [релационни бази данни](https://bg.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8), MongoDB съхранява структурираната информация в [JSON](https://bg.wikipedia.org/wiki/JSON) формат с динамични схеми. Това прави интегрирането на информацията в определени приложения доста по-лесно и по-бързо.

Нерелационната база данни (на [английски](https://bg.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8_%D0%B5%D0%B7%D0%B8%D0%BA): Not only Structured Query Language, NoSQL) предоставя механизъм за съхранение и възстановяване на данни, който използва свободен [съгласуван модел](https://bg.wikipedia.org/w/index.php?title=%D0%A1%D1%8A%D0%B3%D0%BB%D0%B0%D1%81%D1%83%D0%B2%D0%B0%D0%BD_%D0%BC%D0%BE%D0%B4%D0%B5%D0%BB&action=edit&redlink=1) за разлика от по–често ползваната [релационна база данни](https://bg.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BB%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D0%B8). Ползите на този подход включват изчистен дизайн, [хоризонтално мащабиране](https://bg.wikipedia.org/w/index.php?title=%D0%A5%D0%BE%D1%80%D0%B8%D0%B7%D0%BE%D0%BD%D1%82%D0%B0%D0%BB%D0%BD%D0%BE_%D0%BC%D0%B0%D1%89%D0%B0%D0%B1%D0%B8%D1%80%D0%B0%D0%BD%D0%B5&action=edit&redlink=1) и фин контрол върху наличнaта информация. Нерелационната база данни е най-често добре оптимизирано хранилище, съдържащо информация от тип [ключ-стойност](https://bg.wikipedia.org/w/index.php?title=%D0%9A%D0%BB%D1%8E%D1%87-%D1%81%D1%82%D0%BE%D0%B9%D0%BD%D0%BE%D1%81%D1%82_%D1%85%D1%80%D0%B0%D0%BD%D0%B8%D0%BB%D0%B8%D1%89%D0%B5&action=edit&redlink=1). Предназначението ѝ е да улесни процесите по възстановяване и добавяне на информация, с цел оптимизиране на производителността в условия на неумишлено забавяне на системата и въвеждане на прекалено големи количества данни.

MongoDB e документно хранилище. Централната концепция, която стои зад документното хранилище е нотацията за „документ“. Докато всяка документно-ориентирана имплементация се различава според детайлите на тази дефиниция, като цяло, всички те приемат, че документите енкапсулират и кодират данните (или информацията) в някакви стандартни формати. Използваните формати са: [XML](https://bg.wikipedia.org/wiki/XML), [YAML](https://bg.wikipedia.org/w/index.php?title=YAML&action=edit&redlink=1), [JSON](https://bg.wikipedia.org/wiki/JSON) както и бинарни формати като [BSON](https://bg.wikipedia.org/w/index.php?title=BSON&action=edit&redlink=1), [PDF](https://bg.wikipedia.org/wiki/PDF) and [Microsoft Office](https://bg.wikipedia.org/wiki/Microsoft_Office) документи ([MS Word](https://bg.wikipedia.org/wiki/MS_Word), [Excel](https://bg.wikipedia.org/wiki/Excel) и др.).

Различните имплементации предлагат различен подход към организирането и групираненето на документите:

* Колекции
* Тагове
* Невидими метаданни
* Йерархия на директориите

Ако се направи аналогия с релационните бази данни, колекциите могат да се възприемат като таблици, а документите могат да се приемат за записи. Нерелационните и релационните база данни все пак има съществена разлика: всеки запис в таблицата при релационните бази данни има еднаква последователност от полета, докато документите в колекцията могат да имат полета, които са напълно различни.

Адресът на документите се представя в базата данни чрез уникален ключ, който идентифицира документа. Една от другите основни характеристики на документно-ориентираните бази данни е, че освен простото търсене по ключ-документ или ключ-стойност, което може да се използва за извличане на документа, базите данни предоставят и потребителски интерфейс или език за заявки, който ще позволи документите да бъдат откривани въз основа на тяхното съдържание.

<https://www.mongodb.com/document-databases> !!!!!

Структурата на базата данни, която ще се използва за Trip Planner е много проста. Състои се от две основни колекции документи – my places, където ще се съхраняват любимите места, и my routes, където ще се съхраняват запазените маршрути.

Тъй като ще се използва нерелационната база данни MongoDB, която представлява колекция от документи, всеки запис ще е във JSON формат. Следва примерна структура на един запис за любимо място:

{

"\_id": {

"$oid": "590ac0f4bd966f6c18d45f66"

},

"coords": [

42.36026,

23.38043

],

"title": "Цари Мали Град - Белчин",

"info": "Античната крепост \"Цари Мали град\" с църковен ансамбъл и оброк, се проучва от 2007г. Обявена е за паметник на културата с национално значение. Извън крепостта се намира култовият комплекс. Там е бил оброка и каменният кръст за \"Св. Спас\" или Възнесение Господне, почитан до 50те години на XXв, една средновековна църква от XVв и останките на два раннохристиянски храма, а под тях руините на езическо светилище. Едно от уникалните неща на обекта е почти 1700г християнски култ на едно място.",

"img": "https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRZW4yE-mQ-dXLs-M1fuMkV8NrGdKNIyhk-KPHuLbls5jRY3loCNg"

}

Полетата в документа са:

* id – уникален идентификационен номер;
* coords – географските координати на мястото;
* img – url адрес на снимка на мястото;
* title – име на мястото;
* info – информация за мястото;

Примерната структура на един документ (запис) за маршрут е:

{

"\_id": {

"$oid": "590894f9c2ef1616bf6f9ab4"

},

"routeEnds": [

{

"lat": 42.6977211,

"lng": 23.3225964

},

{

"lat": 43.4088034,

"lng": 24.6181

}

],

"routeName": "София-Плевен"

}

Тя съдържа следните полета:

* id – уникален идентификационен номер;
* routeEnds – колекция от координати на географски точки, които представляват краищата на маршрута;
* routeName – името на маршрута;

Двете колекции документи са независим, нямат връзка помежду си.

Сложи различни диаграми – на класовете, на обектите, на връзките, на последователностите….

<https://bg.wikipedia.org/wiki/%D0%A1%D0%BE%D1%84%D1%82%D1%83%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0>

<https://bg.wikipedia.org/wiki/NoSQL>