4. RESTful Web Services

REST архитектура

REpresentational State Transfer: Стандартна уеб архитектура, базирана на HTTP; Всеки компонент е ресурс; Ресурсите се достъпват с използване на HTTP методи; Ресурсите се представят в различни формати: text, JSON, XML

REST сървър- Осигурява достъп до ресурси; **REST клиент** - Получава достъп и променя ресурсите

HTTP методи - Основни HTTP методи, използвани в REST базираните архитектури: GET – Четене на ресурс; POST – Създаване на нов ресурс; DELETE – Изтриване ресурс; PUT – Обновяване на ресурс или създаване на нов

RESTful уеб услуги - Уеб услуга, базирана на REST архитектура: Използва HTTP методи за да реализира концепцията на REST архитектурата; Идентифицира се с URI; Осигурява ресурси /JSON/

Видимост на взаимодействието - Способност на компонент да наблюдава или да посредничи при взаимодействието на два други компонента

- Кешът, проксито, защитната стена и др. могат да наблюдават или да участват в протокола, когато протоколът е видим
- HTTP заявките и отговорите са видими

Следните възможности са зависими от видимостта: Кеширане (премахване на кеша при модифицирани ресурси); Оптимистичен контрол на конкурентността; Договаряне на съдържание (избор на представяне); Безопасност и идемпотентност (повторение HTTP заявка)

Постигане на видимост с HTTP

-HTTP взаимодействия без състояние - Всеки HTTP посредник може да разбере значението на всяка заявка и отговор, без да ги свързва с минали или бъдещи заявки и отговори

-Използване на унифициран интерфейс: OPTIONS, GET, HEAD, POST, PUT, DELETE, и TRACE методите оперират върху един единствен ресурс

-Използване на MIME-подобен формат за кодиране - Ясно разделяне на заглавието (видимо) от тялото (невидимо).

Управление на състоянието

- **Запазване на състоянието в URI**:
  - Кодиране на състоянието в URI и включване на състоянието в представянето посредством линкове
  - Клиентът използва URI за взаимодействие с ресурса
  - Избягване на необходимостта от създаване на сесии в паметта на сървъра
- **Запазване на състоянието при съображения за сигурност и поверителност** - Запазване в хранилище и кодиране на референция към състоянието в URI

НТТР Е ПРОТОКОЛ БЕЗ СЪСТОЯНИЕ! ИНТЕРАКТИВНИТЕ ПРИЛОЖЕНИЯ ИЗИСКВАТ КЛИЕНТА ДА ИЗПЪЛНИ НЯКОЛКО СТЪПКИ! СЪРВЪРЪТ ВРЕМЕННО ТРЯБВА ДА ЗАПАЗИ ТЕКУЩАТА СТЪПКА НА КЛИЕНТА ИЗВЪН ПРОТОКОЛА! РЕШЕНИЕ: УПРАВЛЕНИЕ НА СЪСТОЯНИЕТО ПРИ ЗАПАЗВАНЕ НА НАДЕЖДНОСТ МРЕЖОВА ПРОИЗВОДИТЕЛНОСТ И СКАЛИРУЕНОСТ.

Реализиране на безопасни методи- Безопасните методи не се очаква да имат странични ефекти; Безопасните методи реализират

операции за четене; Клиентът изпраща заявка, знаейки, че тя няма да направи промени в състоянието на ресурса: GET; HEAD; OPTIONS

Реализиране на идемпотентни методи - **Идемпотентността гарантира, че ако клиент повтори заявката си, ефектът ще бъде същия както, ако заявката е само една** - Предпазва при мрежови или софтуерни повреди: GET; HEAD; OPTIONS; PUT; DELETE - Идемпотентните методи са аналогични на „set“ методите при програмните езици (Пример: setPrice).- **POST: Не**

Използване на GET за идемпотентно и безопасно получаване на информация. Предпазни мерки:

-Създайте отговор без кеширане (Cache-Control: no-cache)

-Уверете се, че всички странични ефекти са доброкачествени и не променят критични за бизнеса данни

- Реализирайте операциите като такива, които могат да се повтарят без промяна на ефекта (т.е. идемпотентни)

- **Използвайте POST** при следните ситуации: Създаване на нов ресурс; Промяна на един или повече ресурси; Изпълнение на заявки с голям обем входни данни

-Изпълнение на операции, които не са идемпотентни и безопасни, когато други HTTP методи са неподходящи HTTP инструментите интерпретират POST по следния начин:1/ Кешът не кешира отговори от POST методи 2/ „Crowers“ и подобни инструменти не активират автоматично POST заявки 3/Повечето HTTP инструменти не изпращат повторно POST заявки автоматично

PUT се използва за създаване на нови ресурси само, когато клиентът може да управлява част от URI (например да задава URI). В противен случай се използва POST. Препоръки при използване на PUT:

- Ако клиентът създава URI, сървърът трябва да подаде информация на клиента относно организацията на URIs (валидни и невалидни)

-Вземат се под внимание правилата за сигурност и филтриране, конфигурирани на сървъра спрямо шаблоните за URI

-Препоръчително е клиентът да избира URI вместо да създава нов

Използване на POST при асинхронни задачи- **При GET заявка относно статуса се връща отговор, както следва:**

- В процес на обработка: 200 (OK), отговор за статуса на ресурса

-Успешно изпълнение: 303 (See Other), URI на ресурса с резултат от изпълнението на заявката

- Грешка: 200 (OK), информация за причината поради, която ресурсът не е създаден

Използване на DELETE при асинхронни задачи:

-DELETE заявка, изискваща дълго време за изпълнение Решение:1/ Изпращане на DELETE заявка 2/ Сървърът създава нов ресурс за проследяване на статуса на заявката 3/Клиентът използва новия ресурс, за да проследява статуса на заявката **Подобен подход се използва за асинхронно изпълнение на PUT**.

Идентифициране на ресурси - Използване на имена от домейна за проектиране на уеб услугите:

- Анализиране на домейна за идентифициране на съществителни (create, read, update delete)

-Използване на POST, GET, PUT и DELETE методи за реализация на операциите върху ресурсите Пример: 1/ GET: получаване на информация за адрес 2/PUT: обновяване на адрес 3/ DELETE: изтриване на адрес 4/POST: създаване на адрес

Идентифициране на ресурси- Използването само на съществителни от домейна е ограничаващо -CRUD операциите са част от интерфейса

Гранулярност на ресурсите- Критерии за определяне на гранулярност: Ефективност на мрежата; Размер на представянето на ресурса; Удобство за клиента;

Пример: социална мрежа: Моделиране на потребителя като ресурс заедно с активностите, приятелите и последователите му (груба гранулярност); Моделиране на активностите, приятелите и последователите като отделни ресурси (фина гранулярност)

Гранулярност на ресурсите- Ресурсите се моделират от гледна точка на клиента -Не е необходимо да се следва модела на базата от данни или обектния модел;

Фактори, влияещи върху гранулярността: Възможност за кеширане; Честота на използване; Изменчивост

Подобряване на ефективността на клиентите и сървърите- Разделяне на ресурсите, за да се гарантира, че рядко кеширащи се, рядко променящи се или неизменчиви данни са отделени от често кеширащи се, често променящите се или изменчиви данните

Организиране на ресурсите в колекции

Предимства на колекциите: Реферирane на група от ресурси като един ресурс; Изпълнение на заявки върху колекции; Използване на колекция като „фабрика“ за създаване на нови ресурси;

Критериите за групиране на ресурси в колекции са специфични за приложението: Споделяне на обща база от данни; Наличие на еднакви атрибути; Свойства, който изглеждат сходни за клиента

Организиране на ресурсите в колекции:1/ Колекциите не винаги са йерархични - Пример: Ресурс „user” може да е част от колекции „users”, “friends” и “followers”; **Приложение на колекциите**: Страниране на изгледи в интерфейса на клиента; Търсене в колекция или филтриране по критерии; Създаване на нов член на колекцията посредством изпращане на POST заявка към нея; Еднократно изпълнение на операция върху множество от ресурси

Съставни ресурси

-Съставните ресурси комбинират информация от други ресурси: Пример: потребител с лични данни, колекция от поръчки и колекция от изчакващи оферти; Предимство: Изпращане на една заявка от клиента, вместо няколко последователни за получаване на цялата информация

-Фактори, влияещи върху избора за създаване на съставни ресурси: Честота на заявките за съставни ресурси (при ниска честота се препоръчва използване на кеш, а не на съставен ресурс); **Цена на комуникацията** между клиента и сървъра и между сървъра и бизнес слоя на приложението или базата от данни

Проектиране на изчислителни функции- Изчислителните функции се разглеждат като ресурси - Използва се GET метод, за да се получи представяне на ресурса като резултат от изчислителна функция

Анотиране на представянето на ресурса - Използване на HTTP заглави блокове:

Content-Type: тип на представянето (application/xml, application/json, text/plain, text/csv, application/pdf); Content-Length: размер в байтове на представянето в тялото на HTTP заявката; Content-Language: език на представянето; Content-MD5: използва се за проверка на

консистентност; Content-Encoding: кодиране на представянето (gzip, compress или deflate); Last-Modified: време на последна промяна на представянето на ресурса от страна на сървъра

Интерпретиране на HTTP заглавните блокове:

-Content-Type- При липса на тип сървърът връща код за грешка 400 (Bad Request)

-Content-Length - При Transfer-Encoding: chunked не се проверява за наличие на този заглавен блок

-Content-Encoding- Обработва се прозрачно от мрежовата библиотека

-Content-Language- Стойността на заглавния блок се запазва за последваща интерпретация

Създаване на нови типове - Стандартните типове (media types) не винаги отговарят на нуждите на приложението. Насоки за създаване на нови типове: - Ако типът е базиран на XML се използва подтип, който завършва с +xml; Ако типът е за частно използване, се използва подтип, започващ с vnd (application/vnd.example.org.user+xml); Ако типът е за публично използване, се регистрира в IANA за получаване на RFC Новите типове могат да създадат проблеми с оперативната съвместимост.

Проектиране на JSON представяния- Съдържание на представянето: Self линк към ресурса; Идентификатор за всеки елемент от домейна, включен в ресурса; Ако е наличен локализиран обект в представянето, се включва свойство за идентифициране на езика

Проектиране на URIs- Добри практики: Използвайте домейни и поддомейни за логическо групиране и разделяне на ресурсите; Използвайте дясна наклонена черта (/), за да покажете йерархична зависимост между ресурсите; Използвайте запетайка (,) и точка и

запетайка (;), за да покажете нейерархични елементи; Използвайте малко тире (-) и долна черта (_), за да подобрите четливостта при наличие на дълги сегменти; Използвайте амперсанд (&), за да отделяте параметрите на заявките; Избягвайте използването на разширения (.php, .aspx, .jsp) в URIs

Създаване на заявки- Използва се метод GET -Типове заявки • Filtering: Избор на подмножество от обекти въз основа на критерии; Sorting: Подреждане на резултата от сървъра; Projection: Избор на определени полета от всеки обект, които да бъдат включени в резултата - **Проектиране на URIs за заявки** - Клиентът специфицира критерии за филтриране, полета за сортиране и проекции посредством параметри за заявки (query parameters).

Ad hoc заявки - Клиентът разглежда сървъра като база от данни: Намалена възможност за сървъра да оптимизира хранилището за данни и кеша; Опасност за намаляване на производителността; Тясно свързване на URIs с начина на съхраняване на данните; Препоръка: Избягвайте заявки, които използват езици за заявки с общо предназначение като SQL и XPath

Проектиране на отговори на заявки - Връщане на ресурс като отговор, **представляващ колекция; Връщане на празна колекция, ако заявката не съответства на нито един ресурс –** Пример: Заявка за получаване най-много 5 ревюта на книга, създадени след определена дата, сортирани в обратен хронологичен ред по датата на създаване

Заявки с голям вход (брой параметри) - HTTP не налага ограничения в дължината на URI - Ограничения на браузъра: Chrome: 2MB (2,097,152 символа); Internet Explorer и Edge: 2,083 символа; Mozilla Firefox: няма ограничения, но в адресната лента се

визуализират до 65,536 символа; **Ограничения на уеб сървъра**: IIS: 16,384 символа; Apache: 8,190 символа; Squid: 8,192 символа

Заявки с голям вход- Използване на POST метод поради невъзможност за кодиране на параметри в URI – Пример: Заявка за търсене на работа въз основа на местоположение, квалификация, опит, тип работа, ключови думи, имена на компании и т.н. Кодиране
• application/x-www-form-urlencoded- **Недостатък**: Загуба на възможност за кеширане; Латентност по време на страниране (необходимост от повторно изпълнение на POST заявката)

Съхранени заявки: **Съхранените заявки позволяват кеширане на POST заявки**: Съхраняване на заявката в сървъра; Изпълнение на заявката с GET метод; **Последователност за създаване**:1/ Създава се нов ресурс, чието състояние съдържа критериите на заявката 2/Връщане на отговор с код 201 (Created) 3/Инициализиране на заглавния блок Location с URI на ресурса

Съхранени заявки – Недостатък: Необходимост от постоянно съхраняване на заявки като ресурси; Сървърът може да натрупа голям брой рядко използвани заявки, което води до необходимост от често почистване на кеша; При наличие на голям брой заявки за кеширане, вероятността за попадане в кеша е нисък (кешът може да отхвърли по-рядко използваните URIs)