Основе веб програмирања

Борисав Живановић (borisavz)

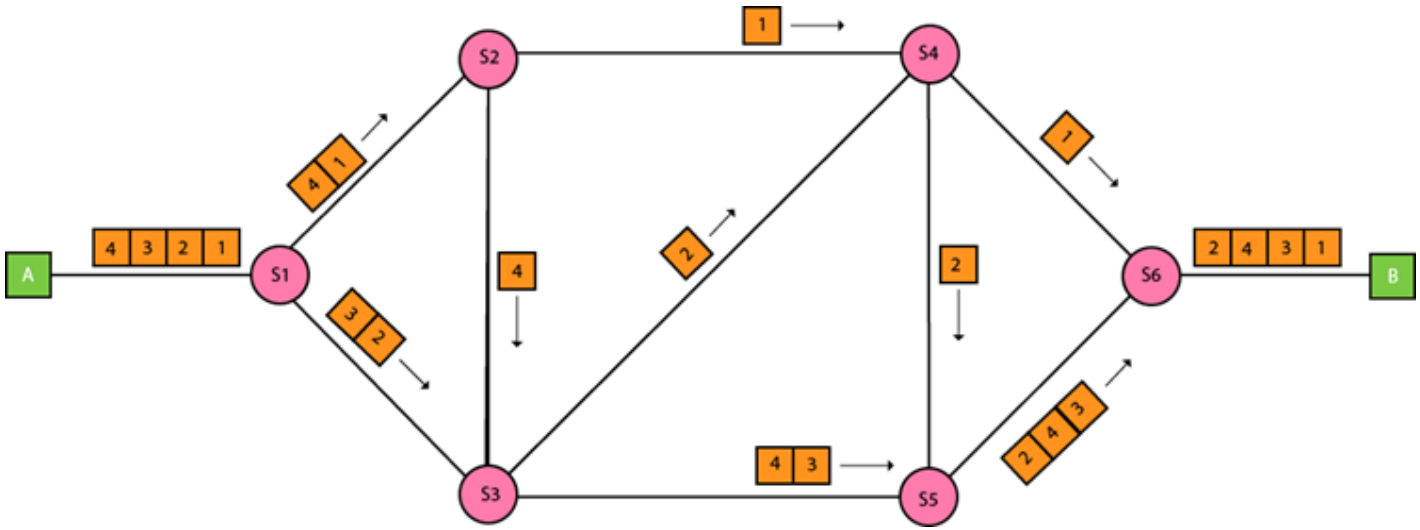
25. јануар 2023.

Садржај

Packet switching I

- Потребно је да поруку пошаљемо примаоцу
- Директна веза са сваким примаоцем није остварива
- Идеја: повезивање пошиљаоца/примаоца у мрежу, дељење комуникационог канала
- Решење: **комутација пакета (packet switching)**
  - Поруку изделимо на пакете
  - Пакетима додамо заглавље (header) са адресом пошиљаоца и примаоца
  - Систем зна путање до примаоца
  - Поруку шаљемо пакет по пакет
  - Само један пакет заузима комуникациони канал
  - Пакети могу да путују различитим путањама кроз мрежу, да дођу у различитом редоследу до примаоца, или да нестану

Packet switching II

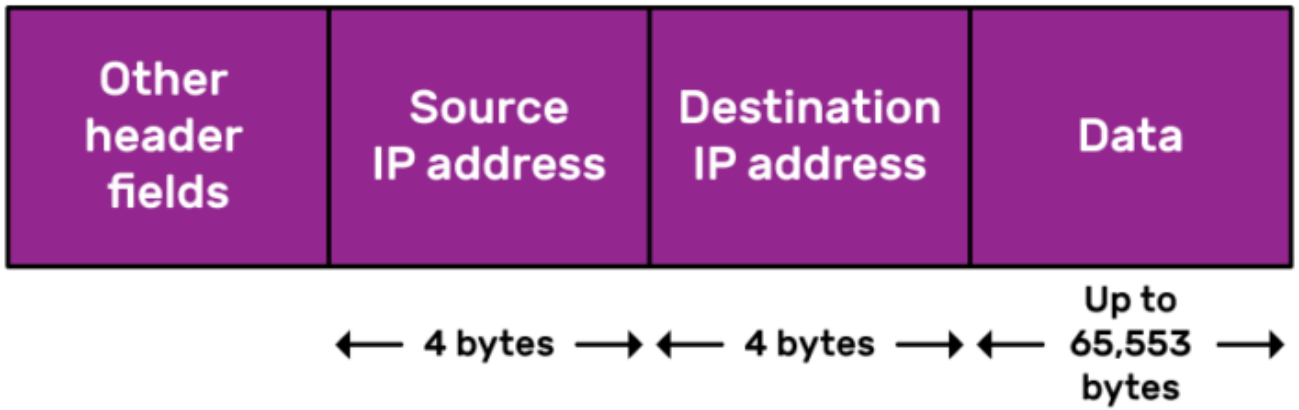


Слика: комутација пакета (packet switching)

Internet Protocol I

- Како би комуницирали у мрежи, потребно је да сваки учесник у комуникацији има додељену **јединствену** адресу
- Поруци придружујемо **заглавље (header)** које садржи:
  - Адресу пошиљаоца (source address)
  - Адресу примаоца (destination address)
  - Додатна поља (верзија IP протокола, flags, TTL, checksum, ...)
- Захваљујући овом заглављу систем зна коме да проследи поруку
- У одговори су адресе пошиљаоца и примаоца **замењене**!

Internet Protocol II

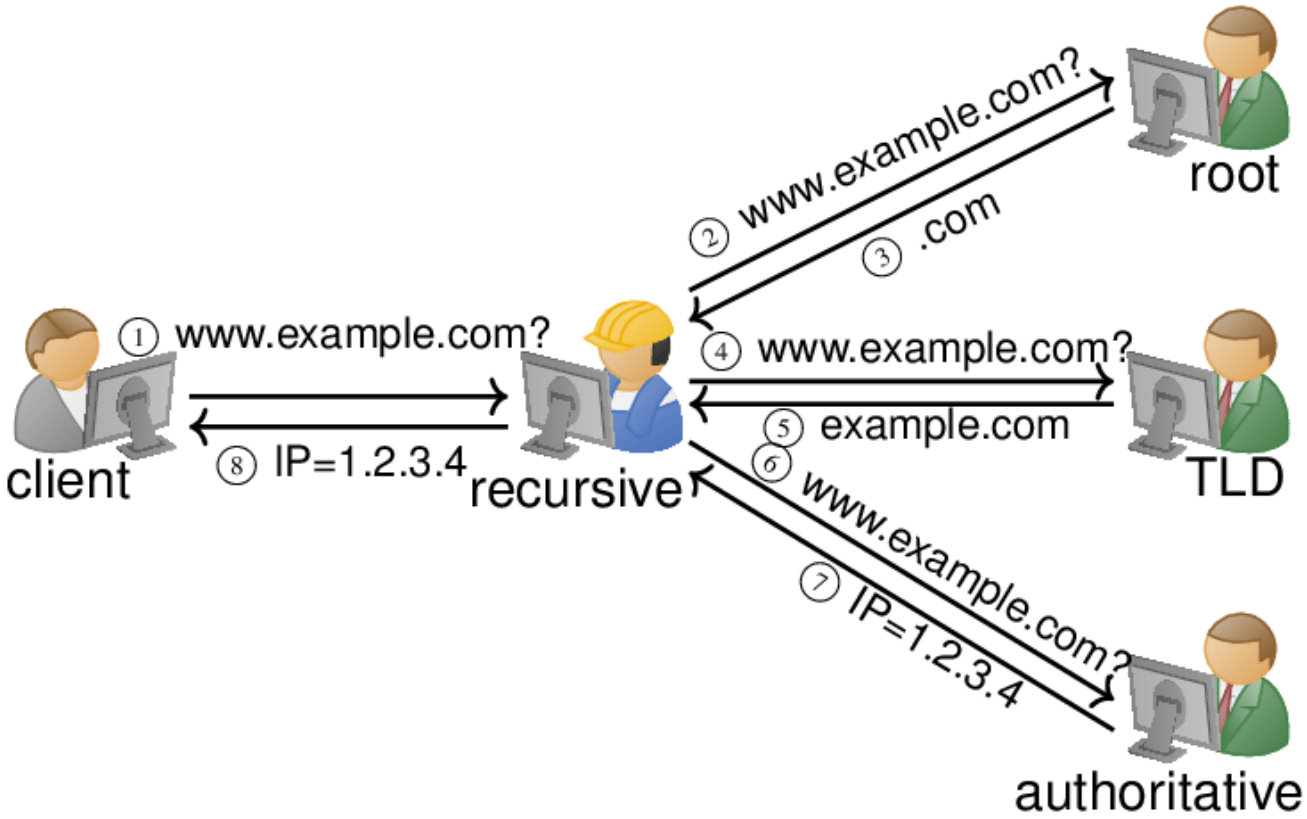


Слика: упрошћена структура IP пакета

DNS I

- Проблем: све више сервера на мрежи
- Није практично памтити сваку адресу у бројчаном облику
- Идеја: систем за придруживање имена, сличан телефонском именику
- Решење: **DNS (Domain Name System)**
  - IP адреси додељујемо симболичко име (домен)
  - Домени су хијерархијски (структура стабла)
  - DNS је одговоран за одређени део хијерархије
  - Као одговор враћа IP адресу или адресу одговорног DNS сервера
  - Морамо знати IP адресу DNS сервера!

DNS II



Слика: DNS упит

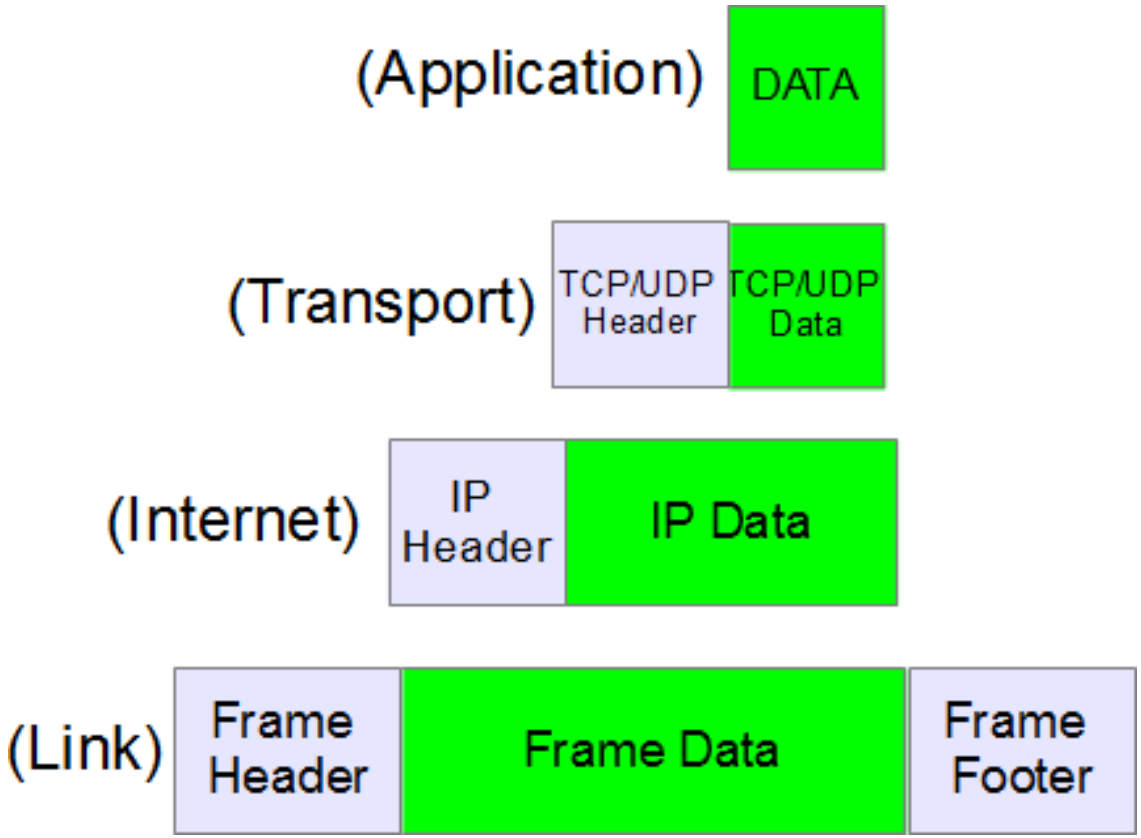
Transmission Control Protocol I

- Решили смо проблем адресирања уређаја на мрежи...
- ...али нисмо проблеме редоследа пристиглих пакета и нестајања пакета
- Додатни проблем: шта ако имамо више мрежних апликација на истом рачунару, како да проследимо поруку одговарајућој апликацији?

Transmission Control Protocol II

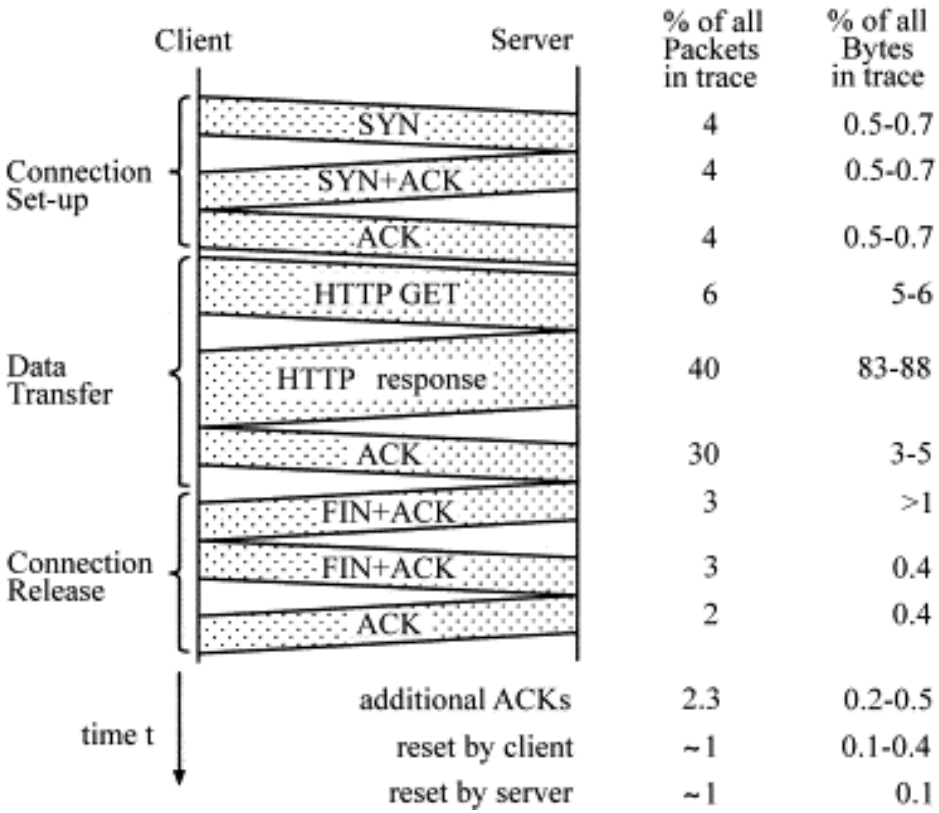
- Решење: **TCP (Transmission Control Protocol)**
  - Додајемо додатно заглавље на нашу поруку
  - Заглавље садржи source и destination port (слично адреси пошиљаоца и примаоца, али се односи на апликацију), sequence number (редослед поруке)
  - Уколико пакет нестане, шаље се поново
  - Оперативни систем осигурава да само једна апликација користи одређени порт

Transmission Control Protocol III



Слика: енкапсулација пакета

Transmission Control Protocol IV

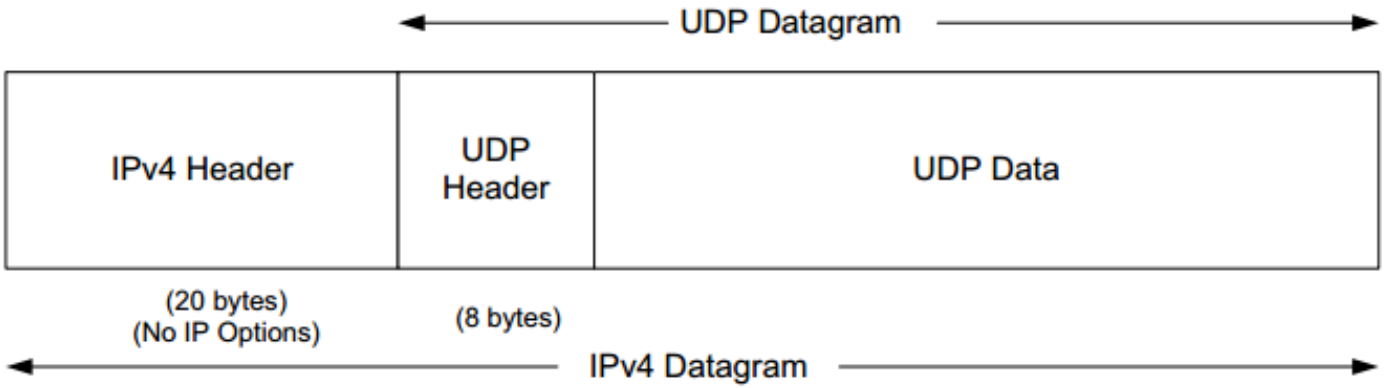


Слика: Ток TCP комуникације

User Datagram Protocol I

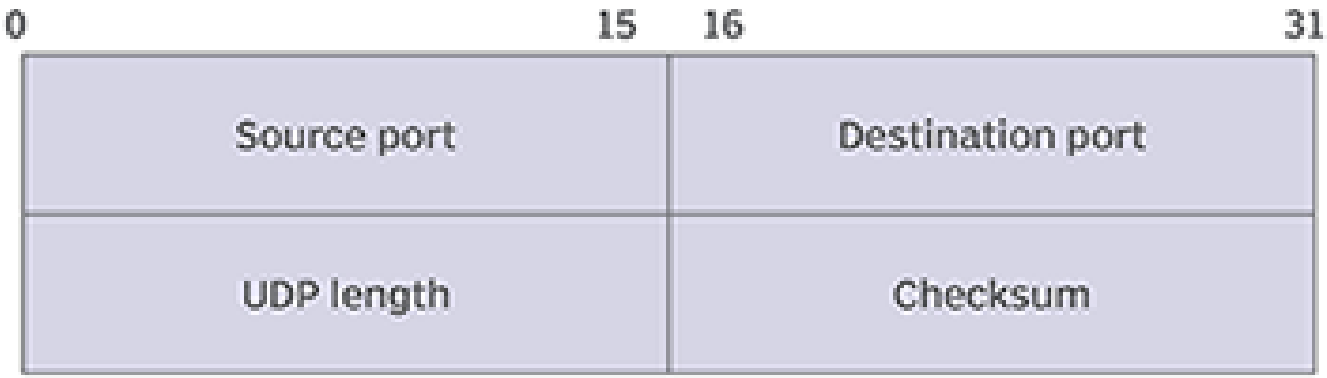
- Успостављање конекције траје одређено време
- За поруке које стају у један пакет, можемо користити једноставнији **UDP (User Datagram Protocol)**
- Задржавамо адресирање апликација, али губимо гаранцију испоруке
- DNS користи UDP
- Омогућава изградњу протокола који имају гаранције испоруке
  - пример: HTTP3/QUIC

User Datagram Protocol II



Слика: енкапсулација пакета

User Datagram Protocol III



Слика: Садржај заглавља

Однос између чворова

- До сада смо говорили искључиво о чворовима који учествују у комуникацији
- Видели смо да један чвор започиње комуникацију, а други даје одговор
- У раду уочавамо две врсте односа између чворова:
  - **peer-to-peer**: обе стране су подједнако важне у комуникацији
  - **client-server**: клијент се обраћа серверу за податке или обављање акције

Клијент-сервер архитектура I

- Модел настао још раних дана рачунарства
- Рачунари су били велики и скупи
- Било је потребно омогућити дељење ресурса између више корисника
- Клијенти су били далеко једноставнији, главна намена је била слање команде и испис резултата
- Данас је овај приступ познат као **thin-client**

Клијент-сервер архитектура II

Слика: PDP-7 (рачунар)

Клијент-сервер архитектура III

Слика: DEC VT100 (терминал)

Клијент-сервер архитектура IV

- Кроз године, рачунарска моћ је расла
- Ово је довело до појаве **PC (Personal Computer)**
  - користи се непосредно
  - без конукурентних корисничких сесија
- Потреба за централним сервером и даље није потпуно избачена, али је могућа далеко већа интерактивност
- Данас је овај приступ познат као **thick-client**
  - пример: Google Docs

Еволуција веб апликација I

- **World Wide Web (WWW)** је изумео Тим Бернерс-Ли у CERN-у
- Оригинална замисао је била систем за дељење докумената
- Језик докумената: **HTML (HyperText Markup Language)**
- Протокол за комуникацију: **HTTP (HyperText Transfer Protocol)**
- Иницијално садржај је био статички (могуће је прегледање искључиво предефинисаних докумената)
- Убрзо су уочени недостаци и настала је потреба за динамичким садржајем

Еволуција веб апликација II

- Идеја: чувати садржај у бази података и на основу њега динамички генерисати HTML документе
- Постоје два решења:
  - **server-side render**: HTML документ генеришемо користећи шаблон и вредности из базе података
  - **client-side render**: са сервера учитавамо основни HTML и JavaScript код, након тога размењујемо JSON објекте

Еволуција веб апликација III

```
<!DOCTYPE html>
<html>

<head>
    <title>Page Title</title>
</head>

<body>
    <h2>Heading Content</h2>
    <p>Paragraph Content</p>
</body>

</html>
```

Слика: HTML документ

Еволуција веб апликација IV

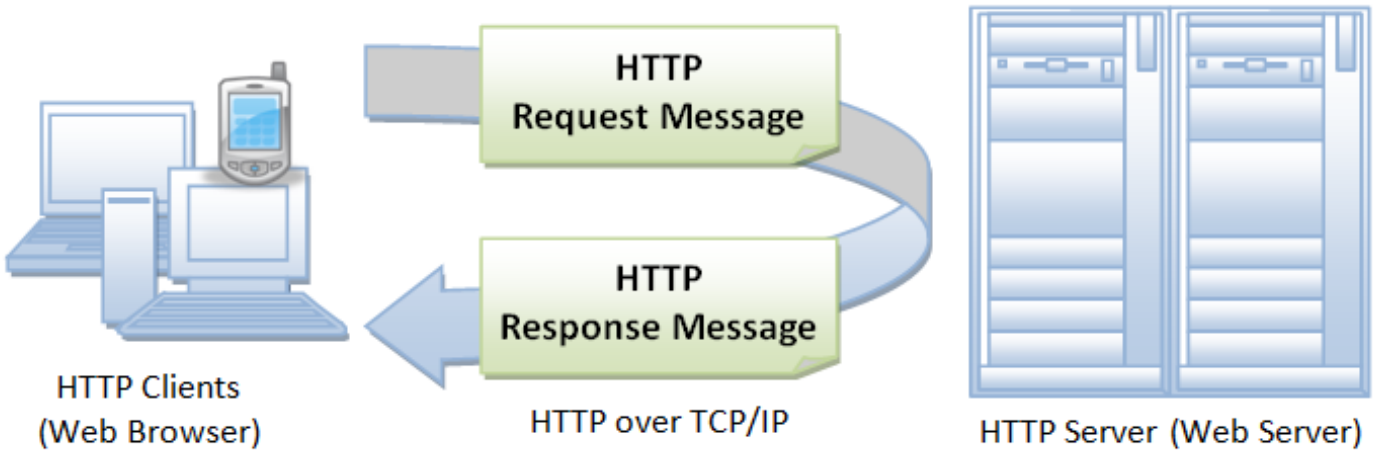
```json
{
  "string": "Hi",
  "number": 2.5,
  "boolean": true,
  "null": null,
  "object": { "name": "Kyle", "age": 24 },
  "array": ["Hello", 5, false, null, { "key": "value", "number": 6 }],
  "arrayOfObjects": [
    { "name": "Jerry", "age": 28 },
    { "name": "Sally", "age": 26 }
  ]
}

```

Слика: JSON објекат

HTTP протокол

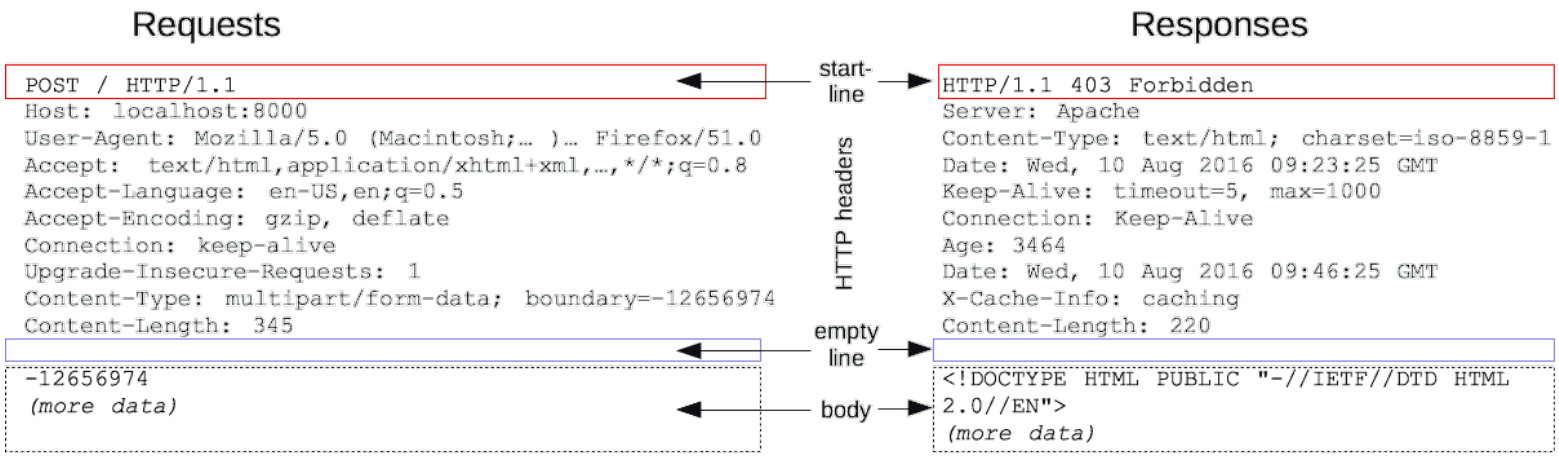
- Текстуални протокол (поруке једноставно могу да читају и људи)
- Користи TCP (гаранција испоруке је неопходна како би протокол успешно функционисао!)
- Подразумевани порт: 80 (HTTP), 443 (HTTPS)
- Stateless протокол
  - неопходно је придружити додатне информације уз сваки захтев како би пратили корисничку сесију
  - обично преко header-a
- Путања означава **ресурс** у систему
  - додатни атрибути кроз **query params**
- Метода означава **акцију** коју желимо да извршимо над ресурсом
- Статусни код означава да ли је акција успешно изврешна, и ако није, разлог

Request-response I



Слика: Request-response модел

Request-response II

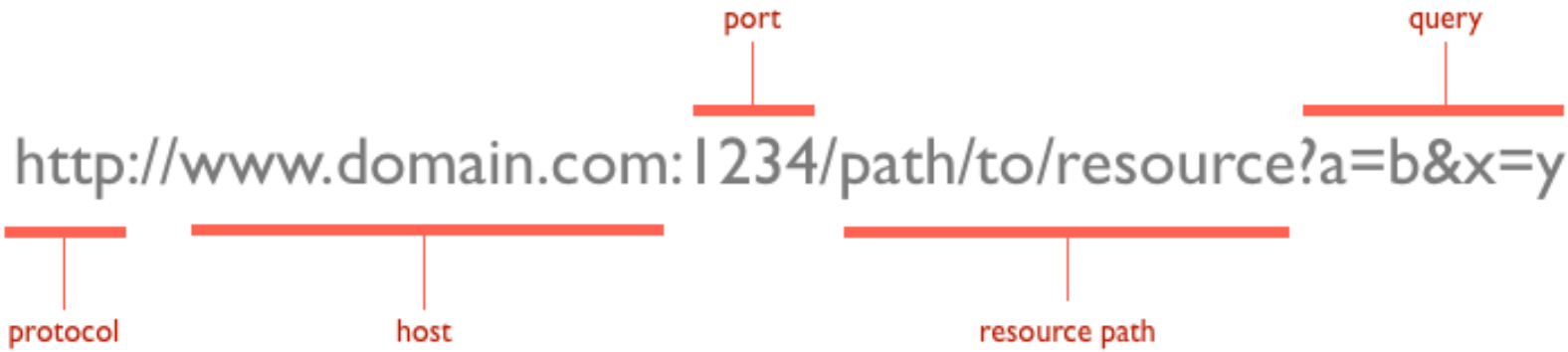


Слика: Садржај **request** и **response** порука

Request-response III

- Формирамо HTTP захтев (string)
- Извршавамо DNS упит како би добили IP адресу сервера
  - могуће је и кеширање DNS одговора на клијентској страни
- Успостављамо TCP конекцију са сервером (подразумевани или наведени порт)
- Захтев шаљемо издељен у пакете
- Чекамо одговор
  - клијенти обично постављају timeout
- Затварамо TCP конекцију
  - потенцијално уско грло уколико у кратком временском периоду шаљемо више захтева
  - исправљено у наредним верзијама протокола

Request-response IV



Слика: URL

HTTP методе

- **GET**: добављање ресурса из система
- **PUT**: измена постојећег ресрса у целости
- **POST**: додавање новог ресурса у систем
- **PATCH**: измена дела постојећег ресурса
- **DELETE**: брисање ресурса из система

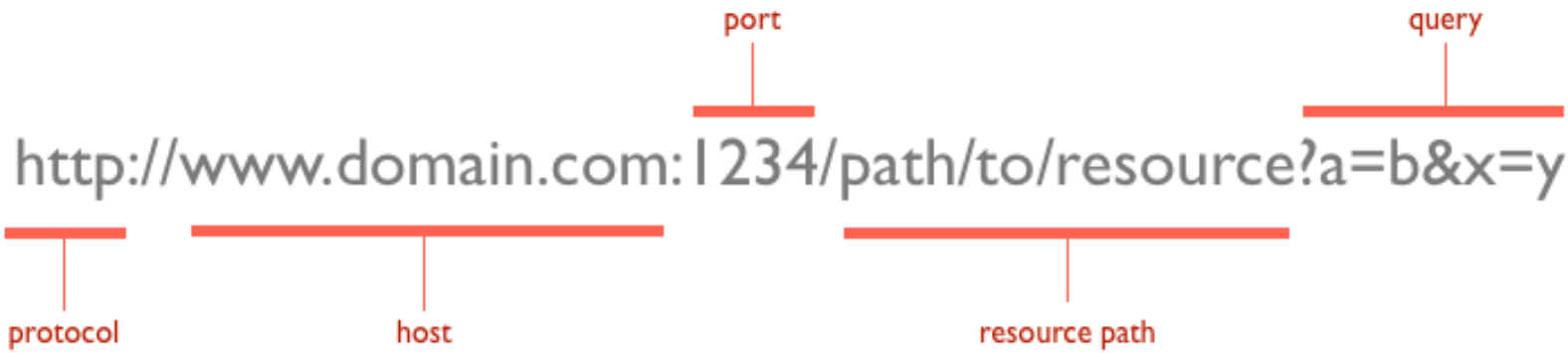
Status codes

- **1xx**: информациони одговор
  - 100 Continue, 101 Switching Protocols, 103 Early Hints, ...
- **2xx**: успешан одговор
  - 200 OK, 201 Created, 202 Accepted, ...
- **3xx**: редирекција
  - 301 Moved Permanently, 302 Found, ...
- **4xx**: грешка клијента
  - 400 Bad Request, 401 Unauthorized, 403 Forbidden, 404 Not Found, 405 Method Not Allowed, 415 Unsupported Media Type, 422 Unprocessable Entity, ...
- **5xx**: грешка сервера
  - 500 Internal Server Error, 501 Not Implemented, 502 Bad Gateway, 503 Service Unavailable, 505 HTTP Version Not Supported, ...

Архитектура веб апликације I

- Потребно је да омогућимо комуникацију преко HTTP
- И да комуницирамо са базом како би извршавали упите
- Једну акцију може да чини више упита ка бази
- Потребно је запис у бази представити структуром података у жељеном програмском језику
- ...и то су, у суштини, компоненте веб апликације

Архитектура веб апликације II



Слика: шематски приказ архитектуре

Controller

- Садржај HTTP захтева претвара у структуру података
- Позива методу из сервисног слоја
- Резултат добијен позивом сервисног слоја претвара у HTTP одговор
- Може да садржи логику за ауторизацију
- Упозорење: грешка коју шаљемо клијенту не сме да открива интерне детаље

Пример имплементације: Go/Gorilla Mux

Пример имплементације: Java/Spring

Service

- Садржи пословну логику апликације
- Једна сервисна метода се састоји из позива једне или више метода из repository
- Уколико база података подржава трансакције, сервисна метода је граница трансакције
  - **commit** уколико је акција успешна
  - **rollback** уколико је акција неуспешна
- Садржи комплетне провере права приступа
  - чест шаблон је да извршимо упит који проверава да ли корисник има право приступа (рецимо, чланство на пројекту), и у зависности од резултата извршимо акцију

Пример имплементације: Go/Gorilla Mux

Пример имплементације: Java/Spring

Repository

- Једна метода представља један упит над базом података
- Параметре прослеђује у упит
  - подсетник: потребно је да се одбранимо од injection напада!
- Резултат упита претвара у одговарајуће структуре података
  - **entity** уколико враћамо записе из базе неизмењене
  - **DTO** уколико упит садржи комплекснија пресликавања (пример: генерисање извештаја)
- У зависности од коришћене базе података/библиотеке, логику за конверзију резултата упита морамо ручно да имплементирамо, или библиотека то чини аутоматски

Пример имплементације: Go/Gorilla Mux

Пример имплементације: Java/Spring

Entity

- Представља записе у бази података
  - додатно: везе ка другим ентитетима
- Омогућује објектно-релационо мапирање
- Може да садржи бизнис логику
  - тема активне дебате

Пример имплементације: Go/Gorilla Mux

Пример имплементације: Java/Spring

Data Transfer Object (DTO)

- Проблем: ентитети потенцијално нису погодни за слање клијенту
- Идеја: применити принцип енкапсулације, трансформација одговора у погодан формат
- Ова компонента је опциона, и често није неопходна
- Могуће је и комбиновање уз entity

Пример имплементације: Go/Gorilla Mux

Пример имплементације: Java/Spring

Middleware

- Често желимо да централизујемо логику која је потребна пре/после извршавања (већине или свих) метода из контролера
  - валидација токена за ауторизацију
  - праћење информација за logging/tracing
- Математички посматрано, одговара композицији функције
- У програмским језицима који имају first-class функције (пример: JavaScript, Go) се имплементира као композиција функција
- Уколико то није подржано, имплементира се механизмом који то опонаша (пример: Java/Aspect Oriented Programming)
- Пресрећемо захтев, прослеђујемо га даље или прекидамо ланац

Пример имплементације: Go/Gorilla Mux

Пример имплементације: Java/Spring