Network2-Uebung

Client-Server-Modell: Anfrage und Antwort

Der Client möchte einen Prozess durchführen und benötigt dafür Informationen und Ressourcen, die auf einem mit ihm verbundenen Server liegen. Also sendet er eine Anfrage (Request) an den Server und fordert diese an. Der Server verarbeitet alle Anfragen, die bei ihm eingehen, und stellt die Elemente bereit (Response), die von ihm gefordert werden. Dabei ist es dem Server überlassen, in welcher Reihenfolge er die Anfragen abarbeitet.

Danach sendet der Server die Ergebnisse seiner Bearbeitung an den Client zurück und dieser kann seinen Prozess durchführen. Der Server identifiziert jeden Client anhand der von ihm gesendeten Adresse (IP-Adresse, Port). Jeder Rechner im Internet hat eine eindeutige IP-Adresse. Die Portnummern 80 (System Ports) ist für sogenannte Webserver reserviert und am meistens benutzt. Dies ermöglicht eine angemessene und spezifische Antwort des Servers an die Clients.

Das Client-Server-Prinzip liegt den meisten Systemen zu Grunde. So gut wie jede Netzwerktechnik basiert darauf. Aber auch die alltäglichsten Dinge, die wir mit einem Computer machen, funktionieren nach dem Client-Server-Prinzip.

Uebung 1:

In diesem Ordner U2 finden wir drei Dateien. Zwei definieren Dateien «Client», die andere ist der Server.
Bei einer neuen Client-Anfrage übermittelt der Client seine Daten an die Transportschicht.
Der TCP stellt dann die Aufteilung der Daten in Pakete sicher und liefert jeweils einen Header für die Vermittlungsschicht. Die Zwischenschicht fügt wiederum einen IP-Header hinzu. Die Netzwerkkarte umgibt zunächst die Daten. Im Empfänger werden die Header entfernt und Schicht für Schicht in umgekehrter Reihenfolge ausgewertet.
Bei einer neuen Client-Anfrage wurde der Socket-Server mit der Port-Nummer 5000 erstellt. TCP-Port 5000 verwendet das Transmission Control Protocol, TCP ist ein verbindungsorientiertes Protokoll, es erfordert Handshaking, um Ende-zu-Ende-Kommunikation einzurichten.

Der Server wartet dann auf die ankommenden Verbindungen und akzeptiert diese, sobald eine Verbindung erkannt wird („`Socket clientSocket = serverSock.accept();`“). Die beiden Variablen, die den Lese- und Schreibfluss in und out steuern, werden zur direkten Verbindung mit dem Sende- und Empfangsfluss initialisiert. Danach wird der generierte Thread ausgeführt:

```
Thread t = new Thread(new ClientHandler(clientSocket));
t.start();
```

Die in der "EinfacherChatClient"-Datei enthaltenen Informationen werden dann aufgerufen. Dies ermöglicht die Anzeige eines Fensters mit einer Beilage, einer "Senden"-Knopf und einer Anzeigetafel mit einer Scroll-Leiste. Die Informationen, die in diesem Fenster eingetragen sind, werden an dem Server gesendet und auf der Konsole erschienen.

Die Dateien "EinfacherChatClientA" funktioniert analog "EinfacherChatClient". Es gibt jedoch einen Unterschied in der Anzeige. Dieser zeigt ein Fenster mit nur einer Beilage an.

Uebung 2
Wir interessieren uns jetzt für die Datei «JavaWebServeur». Dieser baut zunächst einen Socket-Server mit der Port-Nummer 80 auf, der Server wartet auf ankommende Verbindungen und akzeptiert diese, sobald sie erkannt werden (`socket.accept()`). Die beiden Variablen, die den Lese- und Schreibfluss in und out steuern, werden zur direkten Verbindung mit dem Sende- und Empfangsfluss initialisiert. Die Liste von 100 zuvor generierten Threads wird anschliessend

ausgeführt. Der Server sendet dann die Socket-Adresse zurück. Jede Anfrage wird von einem Client-Thread bearbeitet, nach Übermittlung der Antwort schließt den Server den Socket. Die Antworten werden mit einem HTTP/1.0-Header verschickt.
Das Programm gibt « `open browser and enter url: + http://localhost` » an die Konsole zurück, wenn die Verbindung hergestellt werden konnte. Ansonsten wird « `Failed respond to client request:` » angezeigt.
Beim Öffnen der Seite http://localhost stellen wir fest, dass mit der Konsole, die folgenden Text zeigt, eine neue Kundenverbindung aufgebaut wird:

```
New Connection:/0:0:0:0:0:0:0:1
New Connection:/0:0:0:0:0:0:0:1
--- Client request: GET / HTTP/1.1
--- Client request: null
New Connection:/0:0:0:0:0:0:0:1
--- Client request: GET /favicon.ico HTTP/1.1
```

Die Seite zeigt die verschiedenen Informationen, die in HTML programmiert sind:
Der Seitenname: `<title>My Java Web Server</title></head>\n`
«Titel»-Text: `<h1>Welcome to my Java Web Server!</h1>`
Zusammen mit einem Text mit der Serverzeit: `<p>Server Time: " + getCurrentTimeStamp() + "</p>\n"`

Wir haben einen Server-Socket, der mehrere Klienten überwacht. Hier sind zwei Kunden, die gleichzeitig mit dem Server verbunden sind, und jeder Kunde sendet Befehle an den Server. Um diese Verbindungen zu verwalten, wird ein neuer Thread pro Client-Verbindung erzeugt. Während Sie weiterhin nach einem Read suchen, der die Datenströme in jedem Thread blockiert, um die Daten zu verarbeiten. Also wenn der Server jetzt was senden will dann geht er seine Liste durch und schickt die daten an jedem einzelnen Client.

Uebung 3

Wenn man den „index.html“-Datei öffnet, sendet der Computer eine Anfrage in Form der eingegebenen Internetadresse (http-Request) an einen Webserver. Die Website ist fortan über die IP-Adresse des Webservers (hier 172.0.0.1), bzw. über die durch das Domain Name System darauf abgebildete menschenlesbarere Textform (hier https://localhost) erreichbar. Diese Website ist mit Hilfe eines Browsers erreichbar. Der Browser stellt dabei die HTTP-Anfrage an den Server. Der Webserver sendet dann nach der Bearbeitung des Protokolls das Ergebnis in der Form von HTML-Dateien an den Client zurück, welcher jene Dateien dann visualisiert.
Der Browser interpretiert die Anfrage und sendet die Dokumente wiederum per HTTP an den Browser. Der Browser stellt die Dokumente schließlich auf dem Bildschirm dar.
Klickt der Benutzer nun auf den Link Page 2 („`<a href=page2.html>`“) innerhalb dieser Website, so stellt der Browser damit die nächste Anfrage, die vom Server beantwortet werden soll. Der Client entscheidet also, wann welche Daten übertragen werden sollen. Also hier Seite 2 ersetzt die Startseite und man sieht das Bild eines Kätzchens. Man kann mit einem Klick auf den Link „`index.html`“ auf die Hauptseite zurückkehren.

Output der Konsole
Auf der Konsole sieht man folgenden Text:

```
Listening to port 80
1: Incoming call...
2: Incoming call...
< GET /index.html HTTP/1.1
< Host: localhost
< Connection: keep-alive
```

```
< Upgrade-Insecure-Requests: 1
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/87.0.4280.88 Safari/537.36 Edg/87.0.664.66
< Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=
0.8,application/signed-exchange;v=b3;q=0.9
< Sec-Fetch-Site: same-origin
< Sec-Fetch-Mode: navigate
< Sec-Fetch-User: ?1
< Sec-Fetch-Dest: document
< Referer: http://localhost/page2.html
< Accept-Encoding: gzip, deflate, br
< Accept-Language: fr,fr-FR;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6,de;q=0.5
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: text/html
1: Closed.
3: Incoming call...
< GET /page2.html HTTP/1.1
< Host: localhost
< Connection: keep-alive
< Upgrade-Insecure-Requests: 1
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/87.0.4280.88 Safari/537.36 Edg/87.0.664.66
< Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=
0.8,application/signed-exchange;v=b3;q=0.9
< Sec-Fetch-Site: same-origin
< Sec-Fetch-Mode: navigate
< Sec-Fetch-User: ?1
< Sec-Fetch-Dest: document
< Referer: http://localhost/index.html
< Accept-Encoding: gzip, deflate, br
< Accept-Language: fr,fr-FR;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6,de;q=0.5
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: text/html
2: Closed.
< GET /katzenbild.jpg HTTP/1.1
< Host: localhost
< Connection: keep-alive
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/87.0.4280.88 Safari/537.36 Edg/87.0.664.66
< Accept: image/webp,image/apng,image/*,*/*;q=0.8
< Sec-Fetch-Site: same-origin
< Sec-Fetch-Mode: no-cors
< Sec-Fetch-Dest: image
< Referer: http://localhost/page2.html
< Accept-Encoding: gzip, deflate, br
< Accept-Language: fr,fr-FR;q=0.9,en;q=0.8,en-GB;q=0.7,en-US;q=0.6,de;q=0.5
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: image/jpeg
3: Closed.
```

Die erste Zeile zeigt an, dass der Server auf Port 80 hört. Die folgenden Zeilen bedeuten, dass eine Anfrage erfolgt. Die HTTP GET-Methode fordert eine Darstellung der angegebenen Ressource « index.html » an. Der Header der Host-Abfrage gibt an, dass der Host des Servers, an den die Abfrage gesendet wird, localhost ist. Der allgemeine Keep-Alive-Header erlaubt es dem Absender,

darauf hinzuweisen, wie die Verbindung verwendet werden kann, um ein Timeout und eine maximale Anzahl von Anfragen festzulegen. Der HTTP Upgrade-Insecure-Requests Request Header sendet ein Signal an den Server, das die Präferenz des Clients für eine verschlüsselte und authentifizierte Antwort ausdrückt. Mozilla/5.0 ist das allgemeine Token, das sagt, der Browser ist Mozilla-kompatibel. Aus historischen Gründen, fast jeder Browser sendet es heute.
Plattform beschreibt die native Plattform, auf der der Browser läuft (hier Windows). Der Browser ist hier Chrome, und bietet die Version (da "87.0.4280.88").
Der Sec-Fetch-Site fetch-Metadaten-Header zeigt die Beziehung zwischen dem Ursprung eines Request-Initiators und dem Ursprung der Ressource an. Der Metadaten-Header des Sec-Fetch-Modus zeigt den Modus der Anforderung an. Der Sec-Fetch-User fetch-Metadaten-Header gibt an, ob eine Navigationsanfrage durch eine Benutzeraktivierung ausgelöst wurde. Die Navigationsanfrage wurde durch etwas anderes als eine Benutzeraktivierung ausgelöst. Der Sec-Fetch-Dest fetch-Metadaten-Header gibt das Ziel der Anforderung an, so werden die abgerufenen Daten verwendet. Der Referer Request Header enthält die Adresse der anfragenden Seite. Man folgt hier einem Link, der Referer ist also die URL der Seite, die den Link enthält („http://localhost/page2.html“).
Der HTTP-Header der Accept-Encoding-Anfrage gibt an, welche Inhaltskodierung, normalerweise ein Komprimierungsalgorithmus, der Client verstehen kann.
Der HTTP-Header der Accept-Encoding-Anfrage gibt an, welche Inhaltskodierung, normalerweise ein Komprimierungsalgorithmus, der Client verstehen kann. Der Accept-Language-Request-HTTP-Header gibt an, welche Sprachen der Client verstehen kann und welche Locale-Variante bevorzugt wird. (natürliche Sprachen und nicht Programmiersprachen.) Der Server-Header beschreibt die Software, die vom Ursprungsserver verwendet wird, der die Anfrage bearbeitet hat. Es ist also dem Server, der die Antwort generiert hat (man benutzt hier den JavaWebServerAdvanced 0.5). Der Content-Type Entity-Header dient zur Angabe des media-types der Ressource. Der MIME type der Daten ist hier text. Es representiert alle Documente, die Text enthält, der theoretisch von einem Benutzer lesbar ist (Der hier verwendeter Subtyp : text/html).
Schliesslich endet die Verbindung mit der Linie Closed. Eine neue Verbindung wird erzeugt, wenn wir auf den Link auf Seite 2 klicken. Derselbe Prozess wiederholt sich mit den entsprechenden Informationen.

Das Bild befindet sich im "Body" des HTML -Programms. Zur Eingabe eines Bildes wird die Funktion img gefolgt vom Dateipfad /Name verwendet. Man erhält folgende Zeile: `<img src="katzenbild.jpg"></img>`
Innerhalb der Konsole wird festgestellt, dass das Bild auf die gleiche Weise aufgerufen wird wie für Index.html. Die Linien Accept und Content-type ändern sich. Sie sind für ein Bildformat geeignet. Der Referer ist der gleiche wie für die Index.html-Datei.

File 2 Output der Konsole:

```
Listening to port 80
1: Incoming call...
2: Incoming call...
3: Incoming call...
4: Incoming call...
5: Incoming call...
6: Incoming call...
< GET / HTTP/1.1
< Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Upgrade-Insecure-Requests: 1
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Accept-Encoding: gzip, deflate
```

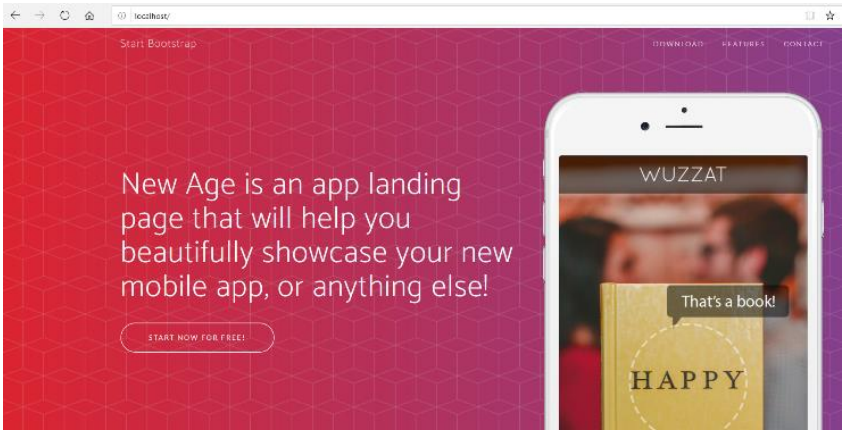
```
< Host: localhost
< Connection: Keep-Alive
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: text/html
1: Closed.
< GET /vendor/bootstrap/css/bootstrap.min.css HTTP/1.1
< Referer: http://localhost/
< Accept: text/css,*/*;q=0.1
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Host: localhost
< Connection: Keep-Alive
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: text/css
< GET /img/google-play-badge.svg HTTP/1.1
< Referer: http://localhost/
< Accept: image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Host: localhost
< Connection: Keep-Alive
< GET /css/new-age.min.css HTTP/1.1
< Referer: http://localhost/
< Accept: text/css,*/*;q=0.1
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< GET /vendor/device-mockups/device-mockups.min.css HTTP/1.1
< Referer: http://localhost/
< Host: localhost
> HTTP/1.0 200 OK
< Accept: text/css,*/*;q=0.1
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< Connection: Keep-Alive
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Host: localhost
> Server: JavaWebServerAdvanced 0.5
< Connection: Keep-Alive
> Content-type: image/svg+xml
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> HTTP/1.0 200 OK
> Content-type: text/css
> Server: JavaWebServerAdvanced 0.5
> Content-type: text/css
5: Closed.
4: Closed.
3: Closed.
< GET /img/demo-screen-1.jpg HTTP/1.1
< Referer: http://localhost/
```

```
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Accept: image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< Host: localhost
< Connection: Keep-Alive
7: Incoming call...
> HTTP/1.0 200 OK
8: Incoming call...
> Server: JavaWebServerAdvanced 0.5
> Content-type: image/jpeg
9: Incoming call...
10: Incoming call...
< GET /vendor/jquery/jquery.min.js HTTP/1.1
< Referer: http://localhost/
< GET /img/app-store-badge.svg HTTP/1.1
< Referer: http://localhost/
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
2: Closed.
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Accept: */*
< Accept: image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< Host: localhost
< Connection: Keep-Alive
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< Host: localhost
< Connection: Keep-Alive
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> HTTP/1.0 200 OK
> Content-type: image/svg+xml
> Server: JavaWebServerAdvanced 0.5
> Content-type: text/javascript
< GET /vendor/bootstrap/js/bootstrap.min.js HTTP/1.1
< Referer: http://localhost/
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Accept: */*
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< Host: localhost
< Connection: Keep-Alive
9: Closed.
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: text/javascript
10: Closed.
8: Closed.
< GET /js/new-age.min.js HTTP/1.1
< Referer: http://localhost/
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Accept: */*
```

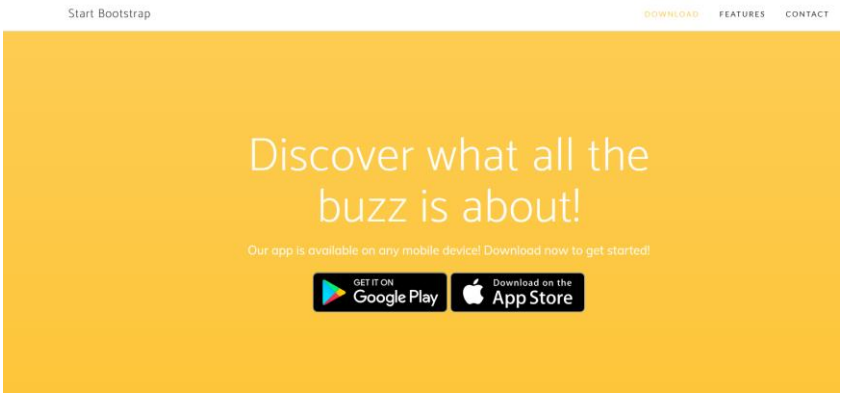
```
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< Host: localhost
< Connection: Keep-Alive
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: text/javascript
7: Closed.
6: Closed.
11: Incoming call...
12: Incoming call...
13: Incoming call...
< GET /img/bg-pattern.png HTTP/1.1
< Referer: http://localhost/
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Accept: image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< Accept-Encoding: gzip, deflate
< Host: localhost
< Connection: Keep-Alive
< GET /img/bg-cta.jpg HTTP/1.1
< Referer: http://localhost/
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Accept: image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
< GET /vendor/device-mockups/iphone_6_plus/iphone_6_plus_white_port.png HTTP/1.1
< Accept-Encoding: gzip, deflate
< Host: localhost
< Connection: Keep-Alive
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: application/octet-stream
< Referer: http://localhost/
> HTTP/1.0 200 OK
< User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/70.0.3538.102 Safari/537.36 Edge/18.18363
< Accept: image/png,image/svg+xml,image/*;q=0.8,*/*;q=0.5
> Server: JavaWebServerAdvanced 0.5
< Accept-Language: fr-FR,fr;q=0.8,de-DE;q=0.5,de;q=0.3
> Content-type: image/jpeg
< Accept-Encoding: gzip, deflate
< Host: localhost
< Connection: Keep-Alive
> HTTP/1.0 200 OK
> Server: JavaWebServerAdvanced 0.5
> Content-type: application/octet-stream
11: Closed.
13: Closed.
12: Closed.
```

Der Ordner files 2 enthält mehr Dateien als der Ordner files. Es ist daher normal, dass die Seite, die auf dem Referer Localhost geöffnet wird, aufwändiger ist. In der Tat ist dies direkt auf der Konsole nach dem Lauschen auf Port 80 mit dem ersten 6 `Incoming call` sichtbar. Die MIME types der Daten sind hier text (Subtyp: `text/html, text/css, text/javascript`), image und application. Application ist Jede Art von Binärdaten, die nicht explizit in einen der anderen Typen fällt; entweder Daten, die in irgendeiner Weise ausgeführt oder interpretiert werden, oder Binärdaten, für deren Verwendung eine bestimmte Anwendung oder Kategorie von Anwendungen erforderlich ist.

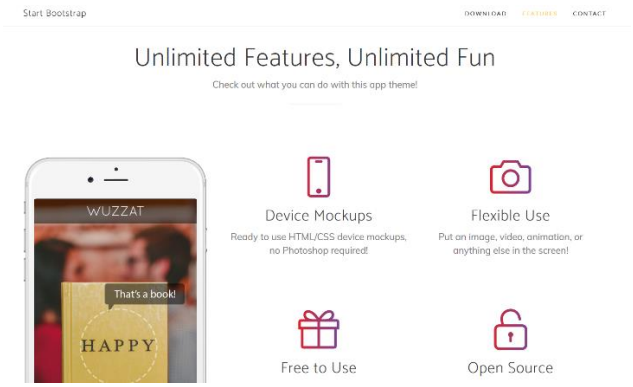
Generische Binärdaten (oder Binärdaten, deren wahrer Typ unbekannt ist) sind `application/octet-stream`. Bild- oder Grafikdaten, die sowohl Bitmap- und Vektor-Standbilder als auch animierte Versionen von Standbildformaten umfassen (Hier verwendeter Subtyp: `image/svg+xml, image/jpeg).` Wenn die Serververbindung hergestellt ist, wird die folgende Startseite angezeigt.

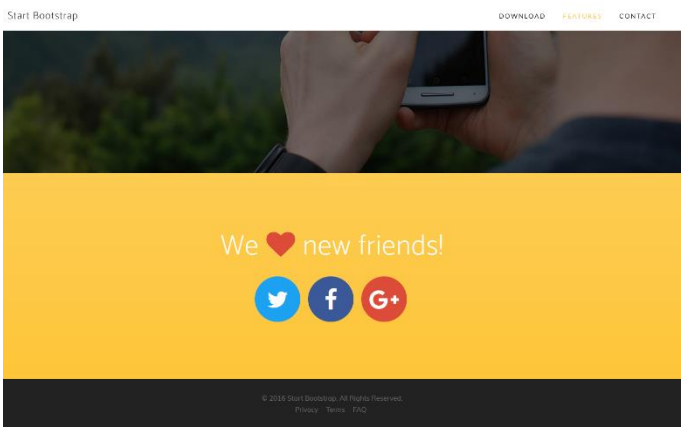


Wenn wir oben rechts auf "Download" klicken, wird die Seite aufgerufen.



Für die Elemente "Features" und "Contact" ist der Vorgang ähnlich, nur dass diese auf zwei verschiedene Seiten verweisen:





Schließlich ist das Element in der linken oberen Ecke gleichbedeutend mit dem Aufrufen der Homepage (Startseite).