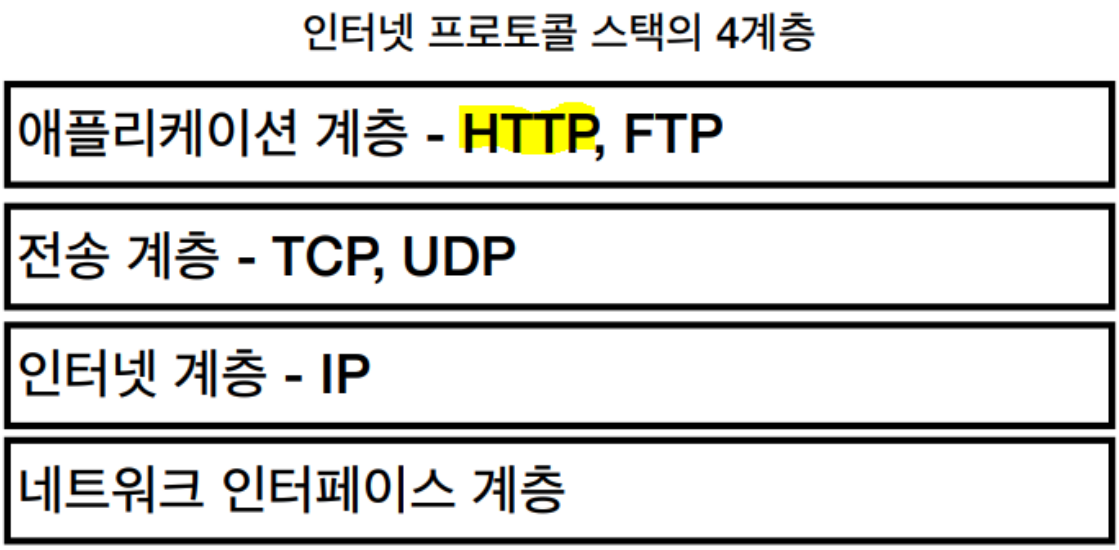
HTTP Methods (1)

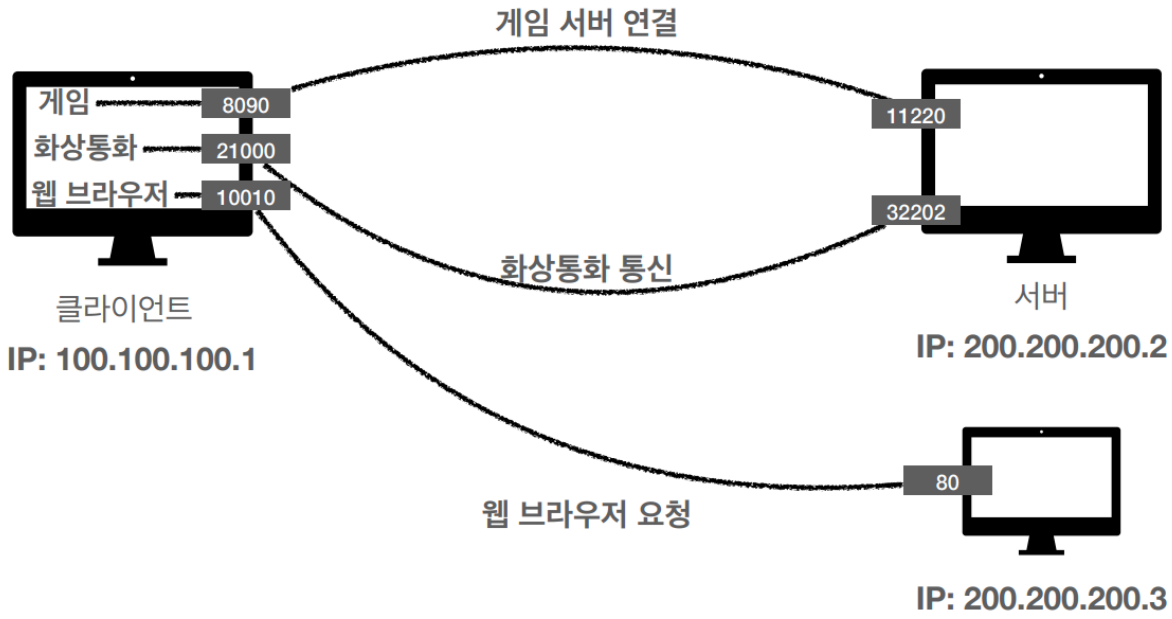
- 백엔드 웹 기술 모두 HTTP 기반으로 구현됨
  - 거의 모든 형태의 데이터 전송 가능 (image, 파일, 영상, JSON, XML(API))
- HTTP API와 REST API 차이
  - https://www.inflearn.com/questions/126743
- Swagger (@ApiOperation 사용): Production 코드에 Swagger코드가 합쳐 보임 * , 실시간 테스트 됨
- REST Docs: 실시간 테스트? , HTTP Client (IntelliJ)

▼ HTTP 관련 기본 개념



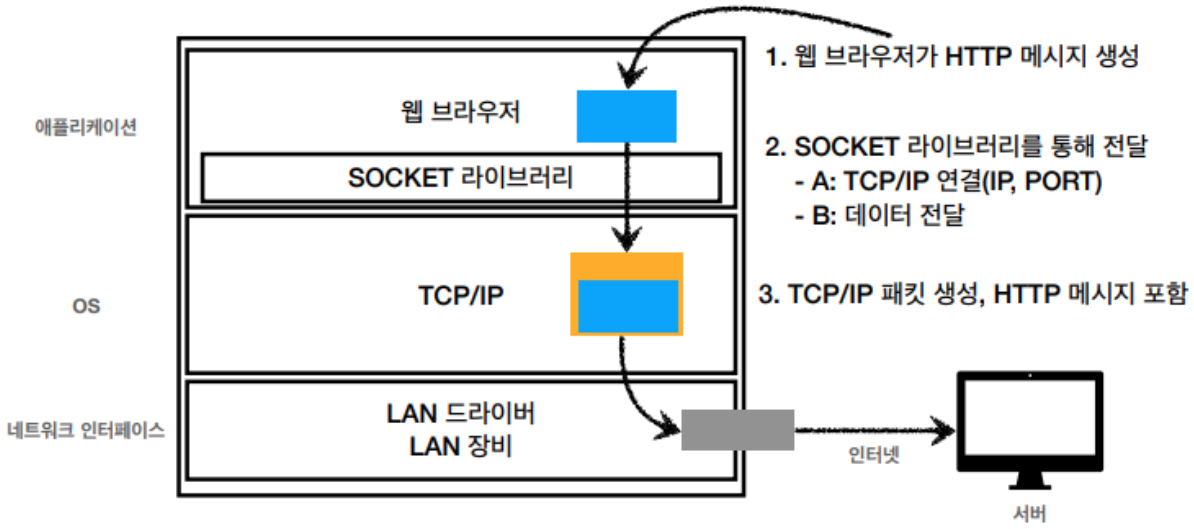
- IP (Internet Protocol)
  - 지정한 IP 주소에 데이터 전달
  - Packet 통신 단위로 데이터 전달
  - 단점: 비연결성, 비신뢰성, 프로그램 구분 안됨 (서버 통신 애플리케이션 여러 개 일 때)
- TCP (Transmission Control Protocol): 신뢰할 수 있는 프로토콜(SYN, ACK)
  - 참고 UDP (User Datagram Protocol): 순서 및 데이터 전달 보증 안되는 대신 단순하고 빠름

- 한번에 애플리케이션 둘 이상 연결해야 할 때 PORT 필요!
  - PORT - 같은 IP 내에서 프로세스 구분
  - IP (아파트), PORT (해당 아파트의 동호수)
  - 0 ~ 65535 할당 가능
  - 0 ~ 1023: 잘 알려진 포트, 사용하지 않는 게 좋음.
  - **HTTP - 80**, HTTPS - 443



- DNS (Domain Name System): IP 기억하기 어려우므로 domain명을 IP 주소로 변환
- URI
  - Uniform: 리소스 식별하는 통일된 방식
  - Resource: 자원, URI로 식별할 수 있는 모든 것(제한 없음)
  - Identifier 다른 항목과 구분하는데 필요한 정
  - https://www.google.com/search?q=ukraine&oq=ukraine&aqs=chrome..69i57j46i512j0i512l8.1593j0j7&sourceid=chrome&ie=UTF-8
  - https://search.naver.com/search.naver?where=nexearch&sm=top_sug.pre&fbm=0&acr=1&acq=선거&qdt=0&ie=utf8&query=선거
  - URL: Uniform Resource Locator
    - URL scheme://[userinfo@]host[:port][/path][?query][#fragment]
    - urserinfo 거의 사용안함
    - port번호도 거의 생략가능
    - 리소스 경로(path), 계층적 구조 설계해야 이해하기 쉬움 /members/100
    - query, key=value 형태. ?로 시작, &로 추가 가능 q=ukraine&..
      - query parameter, query string 등으로 불림, 웹서버에 제공하는 파라미터, 문자 형태
    - fragment: html 내부 북마크 등에 사용, 서버에 전송하는 정보 아님
  - URN: Uniform Resource Name

- 웹브라우저 요청 흐름



▼ HTTP (HyperText Transfer Protocol)

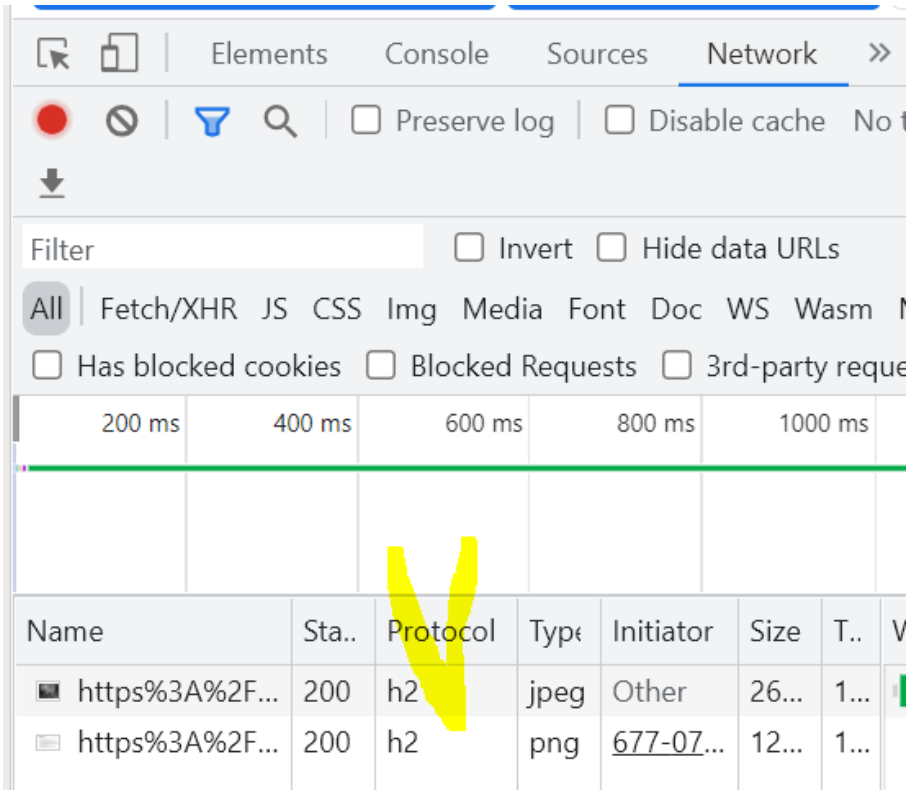
: 인터넷에서 데이터 주고 받을 수 있는 프로토콜 (규칙)

- PUT, PATCH 2010년 쯤 생김.

HTTP 역사

- HTTP/0.9 1991년: GET 메서드만 지원, HTTP 헤더X
- HTTP/1.0 1996년: 메서드, 헤더 추가
- **HTTP/1.1 1997년: 가장 많이 사용, 우리에게 가장 중요한 버전**
  - RFC2068 (1997) -> RFC2616 (1999) -> RFC7230~7235 (2014)
- HTTP/2 2015년: 성능 개선
- HTTP/3 진행중: TCP 대신에 UDP 사용, 성능 개선

- 노션도 h2 사용함 🙂



- HTTP 특징
  - 모든 것이 HTTP
  - Client - Server 구조 (Request - Response)
  - Stateful, Stateless
    - Stateful 상태 유지 → 중간에 점원 바뀌면 무엇을 어떤 지불 방법으로 결제 하는지 모름 (중간에 다른 점원 바뀔 때 상태 정보 다른 점원에게 미리 알려줘야함)
      - 고객: 이 츄러스 얼마예요?
      - 점원: 1,000원 입니다. **(츄러스 상태 유지)**
      - 고객: 5개 살게요.
      - 점원: 5,000원 입니다. 카드, 현금 중에 어떤 걸로 지불하실 거예요? **(츄러스, 5개 상태 유지)**
      - 고객: 카드로 지불할게요.
      - 점원: 5,000원 결제 완료 되었습니다. **(츄러스 5개, 카드 상태 유지)**
    - Sateless 무상태 → 중간에 단른 점원 바뀌어도 됨. **무상태는 응답 서버 쉽게 바꿀 수 있음 → 무한한 서버 증설 가능 (scale out 확장성 좋음)**
      - 고객: 이 츄러스 얼마예요?
      - 점원A: 1,000원 입니다.
      - 고객: **츄러스 5개** 살게요.
      - 점원B: 츄러스 5개는 5,000원 입니다. 카드, 현금 중에 어떤 걸로 지불하실 거예요?
      - 고객: **츄러스 5개 카드**로 지불할게요.
      - 점원C: 5,000원 결제 완료 되었습니다.
    - 무상태 실무 한계 (최대한 무상태로 하되, 필요한 경우 상태유지)
      - 모든 것을 무상태로 설계 할 수 있는 경우도 있고 없는 경우도 있음
      - 무상태: 예) 로그인 필요 없는 단순한 서비스 소개 화면
      - 상태 유지: 예) 로그인
      - 로그인한 사용자의 경우 로그인 했다는 상태 서버에 유지
      - 일반적으로 브라우저 쿠키와 서버 세션 등 사용해 상태 유지
      - 상태 유지는 최소한만 사용
  - 비 연결성(connectionless)
    - TCP/IP 연결: 서버는 연결 유지x, 최소한의 자원 사용
    - 지속 연결(Persistence Connections)으로 응답 다하고 종료 비 연결성 한계인 수 많은 자원 다운로드 및 시간 오래 걸리는 점 해결, h2, h3
  - 최대한 Stateless 방식으로 기억하자! (대용량 트래픽)

    - 실무: 첫 페이지는 이벤트 관련 페이지에서 놀게함...ㅎ

- HTTP 메시지
    - status-line = HTTP-version SP status-code SP reason-phrase CRLF

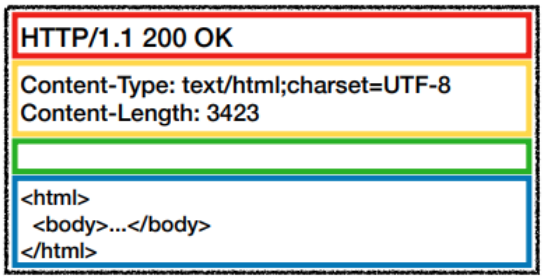
3. Message Format

All HTTP/1.1 messages consist of a start-line followed by a sequence of octets in a format similar to the Internet Message Format [RFC5322]: zero or more header fields (collectively referred to as the "headers" or the "header section"), an empty line indicating the end of the header section, and an optional message body.

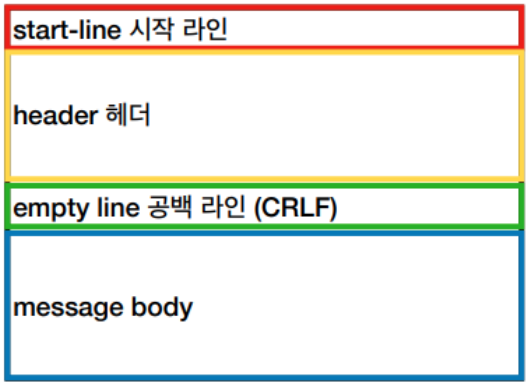
```
HTTP-message   = start-line
                 *( header-field CRLF )
                 CRLF
                 [ message-body ]
```

예) HTTP 요청 메시지

요청 메시지도 body 본문을 가질 수 있음



예) HTTP 응답 메시지



HTTP 메시지 구조

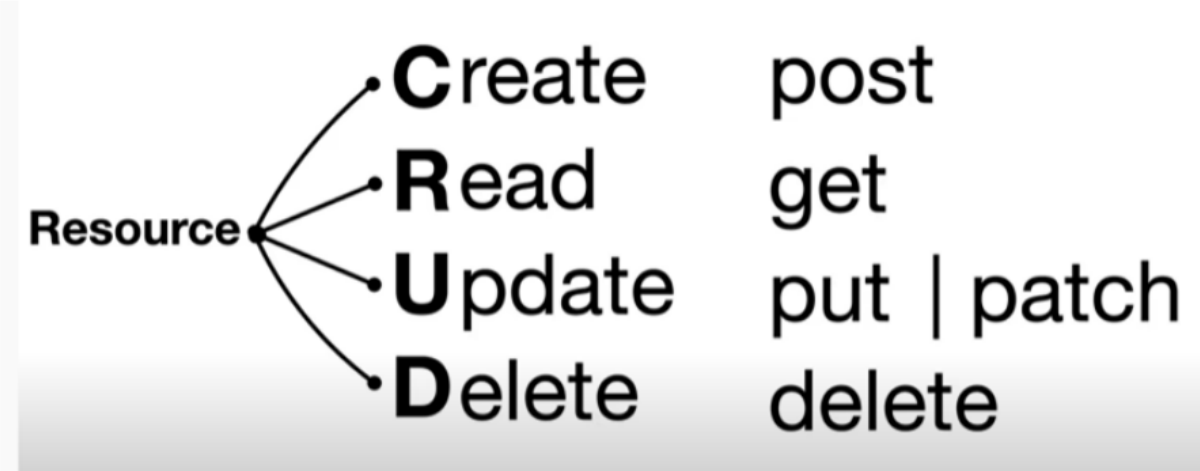
▼ HTTP Methods

- API 설계
    - 리소스와 행위 분리할 것!
    - **URI는 리소스만 식별**

        - 리소스: 상품
        - 행위: 조회, 등록, 삭제, 변경
    - 상품 목록 가져오기 /products
    - 상품 정보 등록하기 /products
    - 상품 정보 가져오기 /products/{id}
    - 상품 정보 수정하기 /products/{id}
    - 상품 정보 삭제하기 /products/{id}

- 행위(메서드) 구분? (지난번 발표 자료 참고)

HTTP METHOD



- GET
    - 리소스 조회 & 필터, 검색 결과 정렬
    - 데이터 전송시 query (query parameter, query string) 사용
        - https://www.google.com/search?q=crazy+putin&oq=crazy+putin
        - (?data=crazyputin&value=crazyputin)
    - message body로 데이터 전달 가능 하지만 권장 되지 않음
- POST
    - 요청 데이터 처리 및 등록에 사용 (+ 애매한 경우 POST 사용하면 됨)
        - 예) HTML form (GET,POST만 지원) 입력 (회원가입, 상품 주문) * **enctype="multipart/form-data"**

        - editor: img, text 분리하는게 Restful 한 것! (면접용으로 알아둘 것)
        - 게시판 등록, 댓글, 신규 주문 생성, 기존 자원 데이터 추가
    1. 새 리소스 생성 (등록): 서버가 아직 식별하지 않은 새 리소스 생성
    2. 요청 데이터 처리: 단순한 데이터 생성, 변경 넘어 프로세스 처리 (결제 완료-배달 시작-배달완료), POST 결과로 새로운 리소스 생성되지 않는 Control URI (/orders/{orderId}/start-delivery
    3. 다른 메서드 처리하기 애매한 경우: JSON 조회 GET 메서드 사용하기 어려울 때
    - message body 통해 데이터 전달
    - 컬렉션(Collection): 서버가 관리하는 리소스 디렉토리, 서버가 리소스 URI 생성 및 관리 (/products) *Control URI
- PUT
    - 리소스 대체, 해당 리소스 없으면 생성 (덮어씀)
    - Client 리소스 위치 알고 있는 경우 /products/100 미리 지정함 (POST와 큰 차이!)
        - products/1 의 **productname: “churros”, status: “in stock (재고 있음)” → status: “out of stock (재고 없음)”** 완전히 대체(productname 삭제됨)
    - message body 통해 데이터 전달
    - 스토어: 파일 /files/star.jpg
- PATCH
    - 리소스 부분 변경
    - prodcuts/1의 productname: “churros”, status: “**in stock (재고 있음)**” → productname: “churros”, **status: “out of stock (재고 없음)**” 상태만 부분 대체
    - PATCH 적용 안되는 경우 POST 사용하면 됨.
    - message body 통해 데이터 전달
- DELETE
    - 리소스 삭제
- 기타: HEAD, OPTIONS, CONNECT, TRACE (참고)
-

참고하면 좋은 URI 설계 개념

- **문서(document)**
  - 단일 개념(파일 하나, 객체 인스턴스, 데이터베이스 row)
  - 예) /members/100, /files/star.jpg
- **컬렉션(collection)**
  - 서버가 관리하는 리소스 디렉터리
  - 서버가 리소스의 URI를 생성하고 관리
  - 예) /members
- **스토어(store)**
  - 클라이언트가 관리하는 자원 저장소
  - 클라이언트가 리소스의 URI를 알고 관리
  - 예) /files
- **컨트롤러(controller), 컨트롤 URI**
  - 문서, 컬렉션, 스토어로 해결하기 어려운 추가 프로세스 실행
  - 동사를 직접 사용
  - 예) /members/{id}/delete

- HTTP 메서드 속성

HTTP 메소드	RFC	요청에 Body가 있음	응답에 Body가 있음	안전	멱등(Idempotent)	캐시 가능
GET	RFC 7231	아니오	예	예	예	예
HEAD	RFC 7231	아니오	아니오	예	예	예
POST	RFC 7231	예	예	아니오	아니오	예
PUT	RFC 7231	예	예	아니오	예	아니오
DELETE	RFC 7231	아니오	예	아니오	예	아니오
CONNECT	RFC 7231	예	예	아니오	아니오	아니오
OPTIONS	RFC 7231	선택 사항	예	예	예	아니오
TRACE	RFC 7231	아니오	예	예	예	아니오
PATCH	RFC 5789	예	예	아니오	아니오	예

  - 안전: 호출해도 리소스 변경 안됨.
  - 멱등 (Indempotent): 몇번 호출하든 결과 똑같음
    - GET, **PUT (결과 대체: 생성하거나 덮어버리거나 같은 결과라서?)**, DELETE

        - (PUT: 나머지 데이터 null)
    - POST는 멱등 아님 * 결제시 두 번 호출하면 같은 결제 중복 발생
    - 자동 복구 매커니즘
    - 멱등은 외부 요인으로 중간에 변경된 리소스 고려하지 않음.
  - 캐시가능 (Cacheable)
    - GET, HEAD, POST, PATCH 캐시 가능하나, GET, HEAD만 캐시로 사용
- Client → Server로 데이터 전송
  - Query Parameter, Message Body 로 전송
    - 정적, 동적(검색, 정렬) 데이터 조회
    - HTML Form 통한 데이터 전송 (GET, POST)
    - HTTP API 통한 데이터 전송: 서버 to 서버, 앱(아이폰, 안드로이드)/웹 (JS 이용 AJAX), Content-Type: application/json

https://www.restapitutorial.com/lessons/httpmethods.html

HTTP Verb	CRUD	Entire Collection (e.g. /customers)	Specific Item (e.g. /customers/{id})
POST	Create	201 (Created), 'Location' header with link to /customers/{id} containing new ID.	404 (Not Found), 409 (Conflict) if resource already exists..
GET	Read	200 (OK), list of customers. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single customer. 404 (Not Found), if ID not found or invalid.
PUT	Update/Replace	405 (Method Not Allowed), unless you want to update/replace every resource in the entire collection.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
PATCH	Update/Modify	405 (Method Not Allowed), unless you want to modify the collection itself.	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid.
DELETE	Delete	405 (Method Not Allowed), unless you want to delete the whole collection—not often desirable.	200 (OK). 404 (Not Found), if ID not found or invalid.

▼ References

- 강의: **모든 개발자를 위한 HTTP 웹 기본 지식, 김영한, 인프런**
- 공식 문서: https://datatracker.ietf.org/doc/html/rfc7230
- 기타 블로그
  - https://yalantis.com/blog/how-to-create-a-restful-api/
  - https://loy124.tistory.com/368
