

OPEN API

API

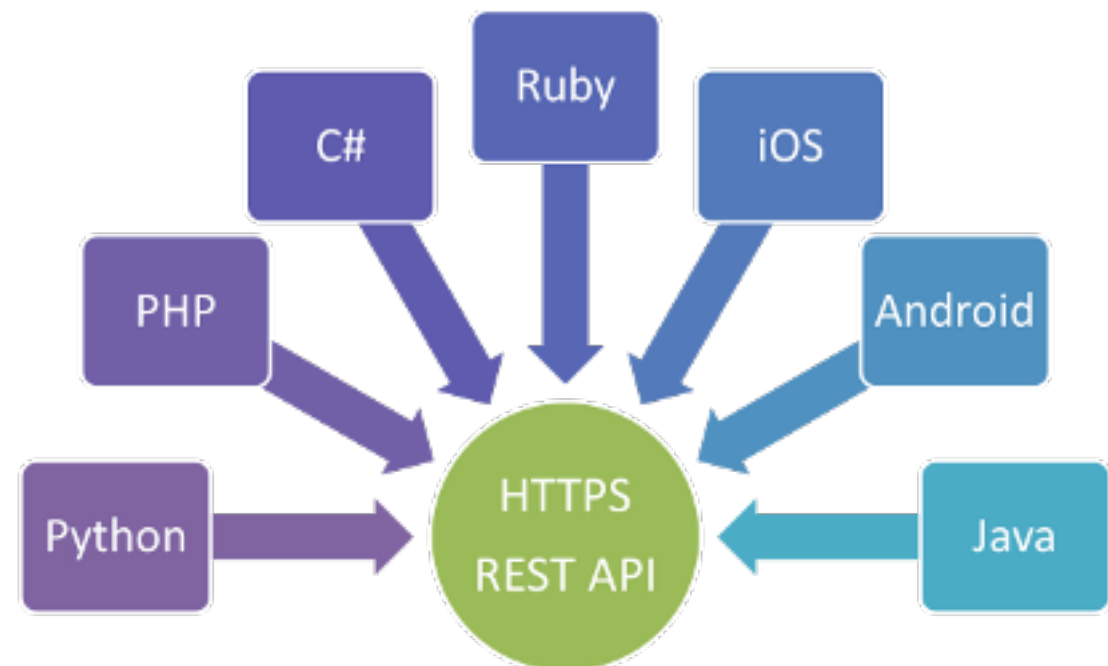
(Application Programming Interface)

- API의 역할



Open API

- Open API는 서비스, 정보, 데이터 등 언제, 어디서나 누구나 쉽게 이용할 수 있도록 개방된 API를 의미 한다.
- 통신망의 구조 및 기술에 독립적으로 새로운 응용 서비스를 쉽게 개발할 수 있도록 한다.
- 주로 웹서비스 형태로 제공된다.



Open API 활용사례

Seoul Bus 2 - 수도권버스정보
Ver. 2.2.2
Database Ver. 20111029

본 어플의 실시간 정보들은 아래의 제공처에서 받아오게됩니다. 따라서, 실시간 정보들의 경우엔 제공처의 서버 사정에 따라 표시되지 않을 수 있습니다. 또한 서울, 인천, 경기 각각 다른 서버를 이용하므로, 이용시 정류소가 통일되지 못한 점을 생각하여주시기 바랍니다.

1.0 테스트에 도움을 주신 분들 >
2.0 테스트에 도움을 주신 분들 >

101 간선버스
도봉,강북,성북,노원권역을 출발, 종로, 중구,용산권역에 도착하는 일련번호 1번 간선버스.

평일
구간 우이동~서소문
1회 운행 정보 37.00km (190분)
1일 운행 횟수 177
배차 간격 4분 ~ 8분
운행 시간 04:00 ~ 23:00

평일

약 3분 16초 (3번째 전) >
7 / 다음: 종로2가

103 약 1분 34초 (2번째 전) >
운행: 05:03~23:32 / 다음: 종로2가

143 도착예정버스 없음 >
운행: 04:22~22:39 / 다음: 종로2가

150 약 1분 35초 (2번째 전) >
운행: 04:41~23:14 / 다음: 종로2가

160 약 1분 32초 (2번째 전) >
운행: 04:37~23:12 / 다음: 종로2가

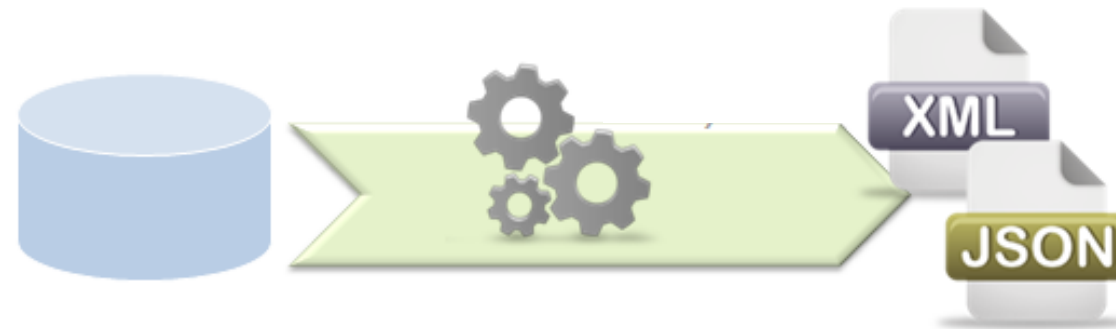
201 약 9분 6초 (5번째 전) >
운행: 04:56~01:19 / 다음: 종로2가

260 약 3분 14초 (3번째 전) >
운행: 04:37~23:10 / 다음: 종로2가

상암DMC홍보관 [14-287] 운행: 04:13 ~ 22:54
상암초등학교 [14-166] 운행: 04:14 ~ 22:56
상암휴먼시아아파트 [14-165] 운행: 04:15 ~ 22:57
월드컵경기장북측 [14-316] 운행: 04:15 ~ 22:57
월드컵경기장서측 [14-106] 운행: 04:16 ~ 22:58
마포농수산물시장월드컵공원

지도 위성 지도+위성

JSON vs XML



<http://localhost:8080/Json/SyncReply/Contacts>

```
{
  - Contacts: [
    - {
      FirstName: "Demis",
      LastName: "Bellot",
      Email: "demis.bellot@gmail.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Jobs",
      Email: "steve@apple.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Ballmer",
      Email: "steve@microsoft.com"
    },
    - {
      FirstName: "Eric",
      LastName: "Schmidt",
      Email: "eric@google.com"
    },
    - {
      FirstName: "Larry",
      LastName: "Ellison",
      Email: "larry@oracle.com"
    }
  ]
}
```

<http://localhost:8080/Xml/SyncReply/Contacts>

```
<ContactsResponse xmlns:i="http://www.w3.org/2003/05/soap-envelope">
  <Contacts>
    <Contact>
      <Email>demis.bellot@gmail.com</Email>
      <FirstName>Demis</FirstName>
      <LastName>Bellot</LastName>
    </Contact>
    <Contact>
      <Email>steve@apple.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Jobs</LastName>
    </Contact>
    <Contact>
      <Email>steve@microsoft.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Ballmer</LastName>
    </Contact>
    <Contact>
      <Email>eric@google.com</Email>
      <FirstName>Eric</FirstName>
      <LastName>Schmidt</LastName>
    </Contact>
    <Contact>
      <Email>larry@oracle.com</Email>
      <FirstName>Larry</FirstName>
      <LastName>Ellison</LastName>
    </Contact>
  </Contacts>
</ContactsResponse>
```

XML의 장단점

```
<ContactsResponse xmlns:i="http://www.w3.org/20
  <Contacts>
    <Contact>
      <Email>demis.bellot@gmail.com</Email>
      <FirstName>Demis</FirstName>
      <LastName>Bellot</LastName>
    </Contact>
    <Contact>
      <Email>steve@apple.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Jobs</LastName>
    </Contact>
    <Contact>
      <Email>steve@microsoft.com</Email>
      <FirstName>Steve</FirstName>
      <LastName>Ballmer</LastName>
    </Contact>
    <Contact>
      <Email>eric@google.com</Email>
      <FirstName>Eric</FirstName>
      <LastName>Schmidt</LastName>
    </Contact>
    <Contact>
      <Email>larry@oracle.com</Email>
      <FirstName>Larry</FirstName>
      <LastName>Ellison</LastName>
    </Contact>
  </Contacts>
</ContactsResponse>
```

- 장점
 - 작성하기가 간편하다(tag구조)
 - XML 사람이 읽기가 쉽다. (즉 각 장보들이 의미하는 바를 한눈에 보기가 좋다.
 - DTD 등 XML자체의 기능을 확장할 여지가 많이 있다.
- 단점
 - 문서의 양이 필요이상으로 많다.
 - 배열형식이나 반복구조의 경우 불필요한 데이터가 계속 해서 나타난다.

JSON 의 장단점

```
{
  - Contacts: [
    - {
      FirstName: "Demis",
      LastName: "Bellot",
      Email: "demis.bellot@gmail.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Jobs",
      Email: "steve@apple.com"
    },
    - {
      FirstName: "Steve",
      LastName: "Ballmer",
      Email: "steve@microsoft.com"
    },
    - {
      FirstName: "Eric",
      LastName: "Schmidt",
      Email: "eric@google.com"
    },
    - {
      FirstName: "Larry",
      LastName: "Ellison",
      Email: "larry@oracle.com"
    }
  ]
}
```

- 장점

- 내용이 함축적으로 최소한의 정보만을 가지고있다.
- 그렇기때문에 XML대비 용량이 획기적으로 줄어들고 속도는 그만큼 빨라지게된다.
- 파싱이 매우 간편하고 사용하기 쉽다.

- 단점

- 내용이 함축적이다 보니 내용의 의미파악은 힘들 수 있다.
- 대용량급의 데이터 송수신엔 부적합 모습도 있다.

주요 API 소개



- Google Developers Page
- 위치 기반 앱 만들기
- Google Maps API 라이선스

kakao

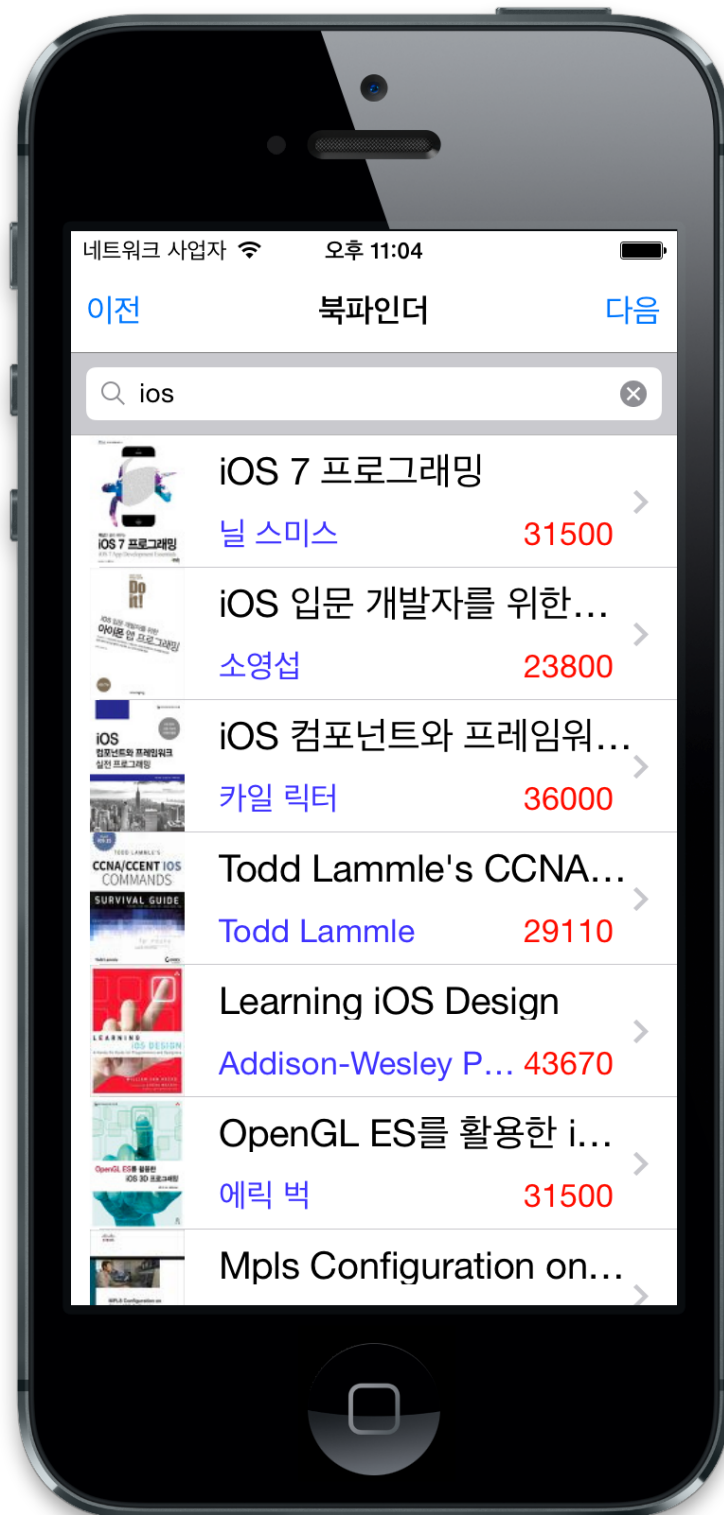
- 카카오 개발자 네트워크
- 다음 검색 API
- 다음 지도 API



- NAVER 개발자센터
- 검색 API
- 지도 API
- 단축 URL API

실습

Daum 검색 API를 이용한 책검색 앱



URLSession

- URLSession은 HTTP/HTTPS 프로토콜을 통해 데이터를 가져오거나 파일을 업로드, 다운로드 하는 기능을 제공하는 클래스
- 앱이 실행 중이지 않거나 일시 중단된 동안 백그라운드 작업을 통해 데이터를 다운로드하는 것도 가능.
- URLSession은 실제 작업을 수행하는 하나 이상의 Task를 생성
- Task의 종류
 - URLSessionDataTask: 서버에 데이터를 검색하는 HTTP GET요청에 사용
 - URLSessionUploadTask: HTTP POST, PUT 매소드를 통해서 디스크에서 웹서버로 파일을 전송시 사용
 - URLSessionDownloadTask: 서버에서 파일을 다운로드할때 사용

URLSession

```
func search(query:String, page:Int){  
    let str = "https://dapi.kakao.com/v3/search/book?query=\(query)&page=\(page)" as NSString  
    guard let strURL = str.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed),  
          let url = URL(string:strURL)  
    else { return }  
  
    var request = URLRequest(url:url)  
  
    request.httpMethod = "GET"  
  
    request.addValue("KakaoAK \(apiKey)", forHTTPHeaderField: "Authorization")  
  
    let session = URLSession.shared  
  
    let task = session.dataTask(with: request, completionHandler:dataTaskHandler)  
  
    task.resume()  
}
```

JSON Parser

```
func dataTaskHandler(data:Data?, response:URLResponse?, error:Error?){  
    guard let data = data else { return }  
    do {  
        if let dic =  
            try JSONSerialization.jsonObject(with: data, options: []) as? [String:Any]{  
                books = dic["documents"] as? [[String:Any]]  
                DispatchQueue.main.async {  
                    self.tableView.reloadData()  
                }  
            }  
        } catch {  
            print(error)  
        }  
    }  
}
```


UI 구성

이전 버튼을 터치했을때

```
@IBAction func actPrev(_ sender: Any) {  
    page -= 1  
    search(query: query, page:page, size:20)  
}
```

다음 버튼을 터치했을때

```
@IBAction func actNext(_ sender: Any) {  
    page += 1  
    search(query: query, page:page, size:20)  
}
```

SearchBar의 서치버튼을 터치했을때

```
func searchBarSearchButtonClicked(_ searchBar: UISearchBar) {  
    page = 1  
    if let q = searchBar.text {  
        query = q  
        search(query: query, page:page, size:20)  
    }  
    searchBar.resignFirstResponder()  
}
```