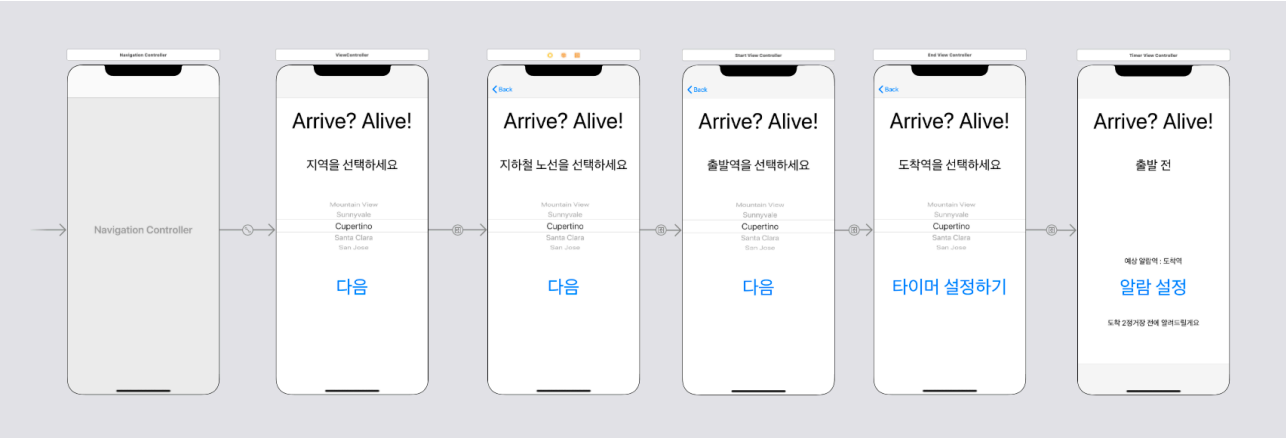
모바일앱프로그래밍2 10조 기술문서

팀명 : 10조 Arrive? Alive!
프로젝트명 : Arrive? Arrive!
소속 : 경북대학교 컴퓨터학부
참여인원 : 김형진, 박주홍, 박효상, 송영욱

클라이언트



[클라이언트 사진 1]

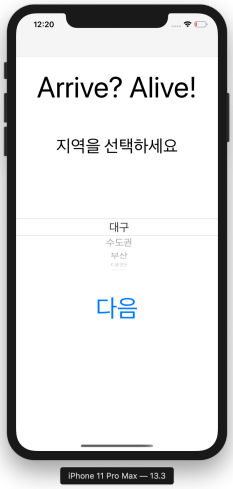
수업시간에 배웠던 Navigation Controller 를 활용하여 전체적인 어플리케이션 시나리오 (클라이언트 사진 1) 를 작성하였습니다. 5개의 ViewController 와 이를 통합하는 1개의 Navigation Controller 를 사용하였습니다. 지역을 선택하는 View 는 Location_ViewController 가 관리하고, 지하철 노선을 선택하는 View 는 Line_ViewController, 출발역을 선택하는 View 는 Start_ViewController, 도착역을 선택하는 View 는 End_ViewController, 타이머를 활용하는 View 는 Timer_ViewController 가 관리합니다.

▼ Supported interface orientations	Array	(1 item)
Item 0	String	Portrait (bottom home button)
▼ Supported interface orientations (i...	Array	(2 items)
Item 0	String	Portrait (bottom home button)
Item 1	String	Portrait (top home button)

[클라이언트 사진 2]

중간발표 중 다른 조 발표에서 했던 내용에 착안해서 세로 모드 (Portrait) 만 지원할 수 있도록 Landscape 는 비활성화 하였습니다. 지하철을 타고 자기 직전에 화면 방향에 따라 버튼이 움직이게 되면 사용자 입장에서 짜증을 느낄 수 있을거라 예상했기 때문에 info.plist 의 Supported interface orientations 내의 Portrait 만 남겨두었습니다 (클라이언트 사진 2).

[클라이언트 사진 3]
Location_ViewController



중간발표까지는 대구에 한정해서 구현하였지만, 이를 확장하여 대구뿐만 아니라 지하철이 설치되어 있는 수도권, 부산, 대전, 광주까지 확장하여 구현하였습니다. 따라서 지역선택 화면을 추가하여 어플리케이션 시작화면으로 설정하였습니다. Swift 의 리스트 (Location_ViewController 내부의 location 변수) 를 활용하여 지역 목록을 저장해두었습니다. Location 변수 내의 내용을 Picker View 기능을 활용하여 출력하였고, 사용자가 스크롤하여 선택할 수 있도록 하였습니다.

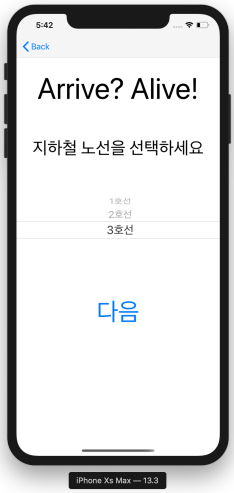
Label 기능을 활용하여 해당 어플리케이션의 이름과 “지역을 선택하세요” 문구를 출력하였습니다.

Button 을 클릭하게 되면 Picker View 에서 사용자가 선택한 내용을 서버로 전송하도록 하였습니다. 서버와의 통신을 위해 IP 와 Host 가 필요한데, 이는 각각 AppDelegate 내의 paramIP 와 paramHost 에 저장되어 있으며 통신이 필요한 ViewController 에서 호출해서 사용할 수 있도록 하였습니다.

동시에 사용자가 선택한 지역이 AppDelegate 내의 paramLocation 에 저장되고, 서버로 보내는 메시지가 만들어져서 보냅니다. 이 때 해당 ViewController 는 “1@대구;” 와 같이 { 1@ + 선택한 지역 + ; } 라는 규칙에 따라서 메시지를 작성합니다.

서버는 해당 메시지를 전송받으면 그 지역에 해당되는 호선들을 클라이언트에게 전송합니다. “1@1호선@2호선@3호선” 과 같이 {1@ + 선택한 지역 내의 지하철 노선들을 @로 구분한 내용} 라는 규칙에 따라서 작성된 메시지를 보내고, 클라이언트는 이를 전송받으면 @를 기준으로 텍스트를 파싱하여 AppDelegate 내의 paramLines 에 리스트 형식으로 저장합니다.

[클라이언트 사진 4]
Line_ViewController



Swift 의 viewWillAppear 기능을 활용하여 화면이 구동되기 전에 AppDelegate 내의 paramLines 내용을 읽어옵니다. 읽어온 paramLines 내용을 Picker View 기능을 활용하여 해당 지역 내의 호선들을 출력하였고, 사용자가 스크롤하여 선택할 수 있도록 하였습니다.

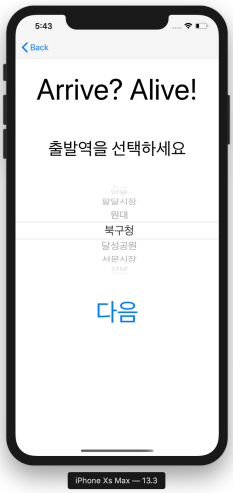
마찬가지로 Label 기능을 활용하여 해당 어플리케이션의 이름과 “지하철 노선을 선택하세요” 문구를 출력하였습니다.

Button 을 클릭하게 되면 AppDelegate 내의 paramIP 와 paramHost 를 호출해서 통신을 할 수 있게 구성됩니다.

동시에 사용자가 선택한 호선이 AppDelegate 내의 paramLine 에 저장되고, 서버로 보내는 메시지가 만들어져서 보냅니다. 이 때 해당 ViewController 는 “2@대구@1호선;” 와 같이 { 1@ + 선택한 지역 + 선택한 노선 + ; } 라는 규칙에 따라서 메시지를 작성합니다.

서버는 해당 메시지를 전송받으면 그 호선에 해당되는 지하철역들을 클라이언트에게 전송합니다. “2@설화명곡@화원@대곡@…@안심” 과 같이 {2@ + 선택한 노선 내의 지하철 역들을 @로 구분한 내용} 라는 규칙에 따라서 작성된 메시지를 보내고, 클라이언트는 이를 전송받으면 @를 기준으로 텍스트를 파싱하여 AppDelegate 내의 paramStation 에 리스트 형식으로 저장합니다.

[클라이언트 사진 5]
Start_ViewController

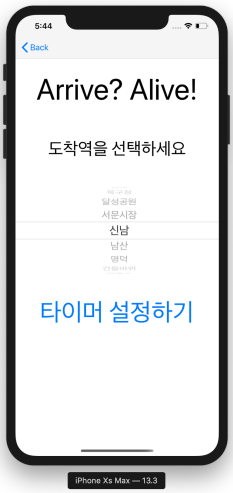


Swift 의 viewWillAppear 기능을 활용하여 화면이 구동되기 전에 AppDelegate 내의 paramStation 내용을 읽어옵니다. 읽어온 paramStation 내용을 Picker View 기능을 활용하여 해당 지역 내의 지하철역들을 출력하였고, 사용자가 스크롤 하여 선택할 수 있도록 하였습니다.

마찬가지로 Label 기능을 활용하여 해당 어플리케이션의 이름과 “출발역을 선택하세요” 문구를 출력하였습니다.

Button 을 클릭하게 되면 이때는 서버 통신이 필요하지 않기 때문에 IP 와 Host 정보를 읽어오지 않습니다. 대신에 사용자가 선택한 출발역이 AppDelegate 내의 paramStart 에 저정됩니다.

[클라이언트 사진 6]
End_ViewController



Swift 의 viewWillAppear 기능을 활용하여 화면이 구동되기 전에 AppDelegate 내의 paramStation 내용을 읽어옵니다. 읽어온 paramStation 내용을 Picker View 기능을 활용하여 해당 지역 내의 지하철역들을 출력하였고, 사용자가 스크롤 하여 선택할 수 있도록 하였습니다.

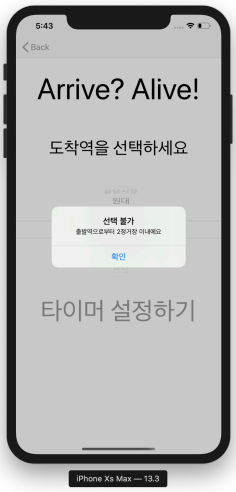
마찬가지로 Label 기능을 활용하여 해당 어플리케이션의 이름과 “도착역을 선택하세요” 문구를 출력하였습니다.

Button 을 클릭하게 되면 AppDelegate 내의 paramIP 와 paramHost 를 호출해서 통신을 할 수 있게 구성됩니다.

동시에 사용자가 선택한 도착역이 AppDelegate 내의 paramEnd 에 저장되고, 서버로 보내는 메시지가 만들어져서 보냅니다. 이 때 해당 ViewController 는 “3@대구@1호선@설화명곡@안심;” 와 같이 { 3@ + 선택한 지역 + 선택한 노선 + 출발역 + 도착역 + ; } 라는 규칙에 따라서 메시지를 작성합니다.

서버는 해당 메시지를 전송받으면 출발역으로부터 도착역까지 2정거장 이내인 경우와 아닌 경우로 구분합니다.

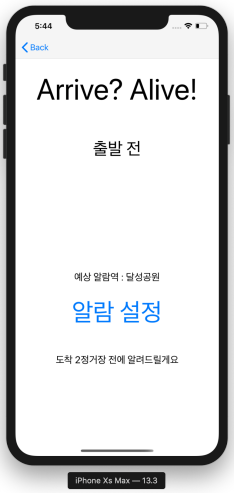
[클라이언트 사진 6-1]
Error Detection



1) 출발역으로부터 도착역까지 2정거장 이내인 경우는 “0@“ 이라는 메시지를 보내고, 클라이언트는 이를 전송받으면 2정거장 이내의 역을 선택했다는 메시지 창을 사용자에게 출력합니다. (클라이언트 사진 6-1) 2정거장 이내에서는 자면 안된다는 생각을 하였기 때문에 다음과 같은 예외처리를 하였으며, 사용자가 잘못 역을 선정한 경우에 한해서 다시 출발역을 재설정하거나 도착역을 재설정하도록 설계하였습니다.

2) 출발역으로부터 도착역까지 2정거장 밖인 경우는 “3@반야월@47” 과 같이 {3@ + 알람 예상역 + 알람 예상역까지 도착하는데 걸리는 시간 (분 단위)} 라는 규칙에 따라서 작성된 메시지를 보내고, 클라이언트는 이를 전송받으면 @를 기준으로 텍스트를 파싱하여 알람 예상역을 AppDelegate 내의 paramAlarmStation 에, 알람 예상역까지 도착하는데 걸리는 시간을 paramTime 에 저장합니다.

[클라이언트 사진 7]
Timer_ViewController



Swift 의 viewWillAppear 기능을 활용하여 화면이 구동되기 전에 AppDelegate 내의 paramAlarmStation 내용과 paramTime을 읽어옵니다.

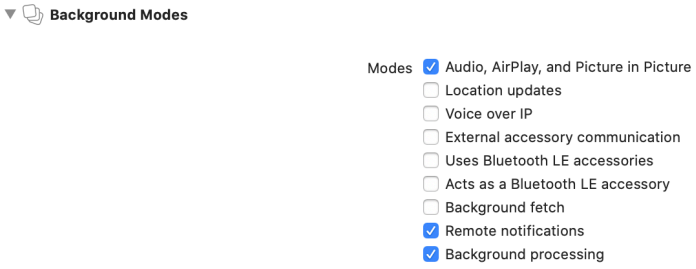
읽어온 내용 중 paramTime 은 분단위로 작성되었기 때문에 60초를 곱한 뒤 time 변수에 저장합니다. 읽어온 내용 중 예상 알람역인 paramAlaramStation 은 AlaramStation_Label 에 “예상 알람역 : paramAlaramStation“ 과 같은 형태로 출력됩니다.

마찬가지로 Label 기능을 활용하여 해당 어플리케이션의 이름을 출력하였습니다. 또한 State 라는 Label 을 추가하여 현재 어플리케이션의 상태를 출력하였습니다. 화면 제일 하단에 있는 Label 은 “도착 2정거장 전에 알려드릴게요” 라는 문구를 출력합니다.

초기 Button 의 내용은 “알람 설정” 으로 되어 있으며, 이를 클릭하면 timeDown 이라는 Label 에 매초마다 time 의 내용이 분단위와 초단위로 구분되도록 계산되어 출력됩니다. 카운트를 하기 위해 scheduledTimer 기능을 활용하였으며 카운드되는 동안에 Button 의 내용은 “알람 설정 완료” 로 변경되며 카운트가 완료될 때까지 비활성화되도록 설정하였습니다. State Label 은 “가는 중” 이라고 출력합니다. (클라이언트 사진 7-1)

[클라이언트 사진 7-1]
알람 설정 완료

[클라이언트 사진 7-2]
Background Mode

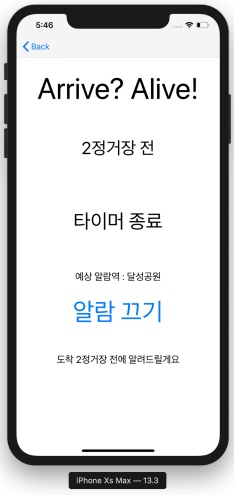


또한 사용자 입장에서 자는 동안에 화면을 끄는 경우가 많을 것으로 예상하여 베터리 소모를 아끼고자 백그라운드에서 동작할 수 있도록 설정하였습니다. (클라이언트 사진 7-2)

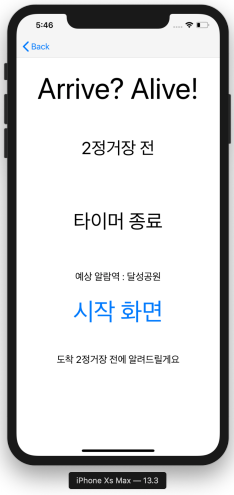
타이머가 완료되면 timeDown Label 은 “타이머 종료” 를 출력하게 하였으며 State Label 은 “2정거장 전” 이라고 출력합니다. 이 때 사용자에게 타이머가 완료되었음을 알리기 위해서 Swift 에서 제공하는 AudioServicesPlayAlertSound 를 사용하여 알람을 알리게 하였으며 Button 을 활성화 시킵니다. 이때 Button 의 내용은 “알람 끄기” 로 변경됩니다. (클라이언트 사진 7-3)

Button 을 클릭하게 되면 알람이 꺼지게 되며, Button 의 내용은 “시작 화면” 으로 변경됩니다. 다시 Button 을 클릭하게 되면 Navigation Controller 의 popToRootViewController 기능을 사용하여 처음 화면인 지역 선택 화면 (클라이언트 사진 3) 으로 돌아오게 됩니다. (클라이언트 사진 7-4)

[클라이언트 사진 7-3]
알람 끄기

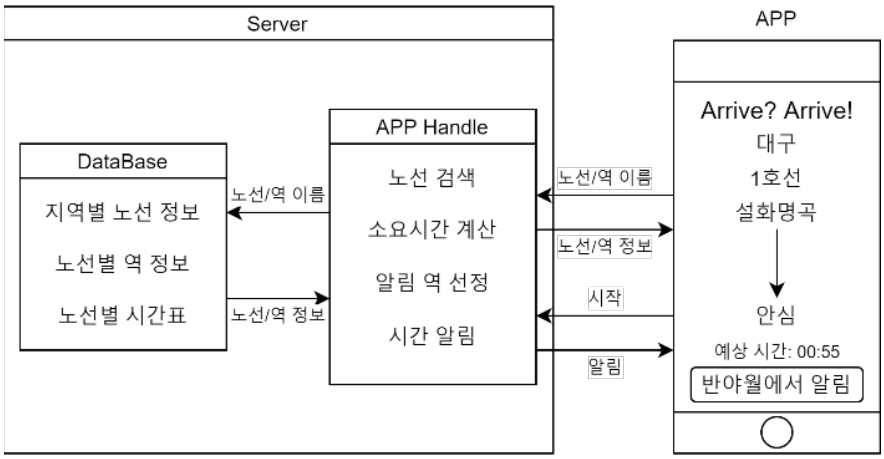


[클라이언트 사진 7-4]
시작 화면



서버

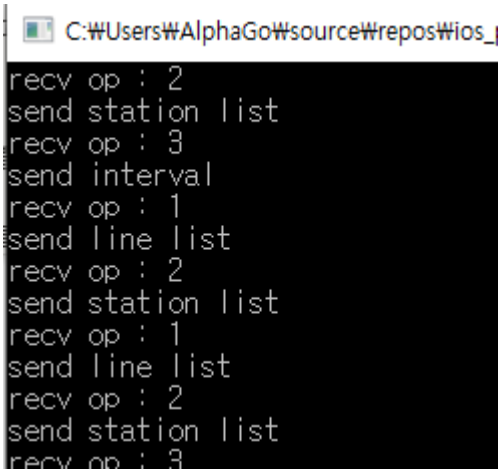
[서버 사진 1]
시스템 다이어그램



어플리케이션에서 모든 데이터를 저장하기에는 메모리 부하가 크기 때문에 서버를 두어 선택한 도시, 선로, 역에 따라서 정보를 전달합니다. 정보 전달 시 예외처리와 계산을 서버에서 감당하고 결과만을 어플리케이션에 전달하여 줍니다.

C++ 기반으로 서버를 작성하였으며 여러 클라이언트 요청을 막힘없이 해결할 수 있도록 접속 클라이언트 마다 쓰레드를 생성하여 헨들링 하고 어플리케이션과의 통신 방법은 TCP 소켓을 통해 결과를 전송합니다. 폴링방식을 이용하여 어플리케이션에서 잘못 선택을 하거나 오류로 인하여 종료 혹은 이전 단계로 돌아가더라도 처리를 할 수 있습니다.

[서버 사진 2]
서버 콘솔 동작



- 동작 코드에 따라 동작하며 코드에 해당하는 행동은 다음과 같습니다.
1. 사용자가 첫 화면에서 지역을 선택하게 되면 해당 지역의 노선 리스트를 소켓을 통하여 전송합니다.
2. 노선을 선택하게 되면 해당 노선에서 갈 수 있는 역 리스트를 전송합니다.
3. 마지막으로 사용자가 출발/도착역을 선택하면 먼저 두 역 사이의 간격이 두 정거장 초과인지 확인을 합니다. 2정거장 초과라면 출발역에서 도착역에서 2정거장 전인 역까지의 걸리는 시간을 계산하고 알람을 울릴 역 이름, 걸리는 시간을 어플리케이션에 전송하게 됩니다. 만약 2 정거장 이하의 역을 선택하게 되면 보통 5분 이하의 시간이 소요되므로 알람을 설정하지 말도록 별도의 메시지를 보내게 됩니다.

데이터베이스에는 지역, 노선, 역 간의 정보가 각각의 테이블에 나누어 저장되어 있고 서버의 부하를 줄이기 위해 1, 2번 동작은 쿼리로 리스트를 불러오고 3번 동작은 sql 사용자 함수를 통하여 계산을 하도록 하였습니다.

[서버 사진 3]
데이터베이스 내용