Smart Navigation Application

Group 16
Neela Balkaran, Evan Dorundo, Michael Kirsche, Clifford Rice, Jason Tiller, and Ryan Zimmerman
Outline

- Concept of Operations
- Project Management Plan
- Software Requirements Specification
- Test Plan
- High-level Design
- Detailed Design
Concept of Operations
Current System

- Currently no existing system designed to solve this direct problem
- Closest alternative is Google Maps - provides user complete control but is time-consuming and can lead to suboptimal routes
Modes of Operation

- Input - user provides list of activities
- Navigation - application provides list of directions
Operational Features

- Interface for entering activities in order in which they should be performed
- Ordered list of locations the user should visit
- Navigation between consecutive locations
- Ability to detect user’s location as a starting point of the navigation
- Different directions based on user’s mode of transportation (driving, walking, etc.)
Analysis

● Improvements
  ○ Shorter travel distance
  ○ Less time making decisions

● Disadvantages
  ○ Battery Usage

● Limitations
  ○ Requires smartphone, GPS, and Internet
  ○ Decisions not based on user’s preferences
Analysis (continued)

● Risks
  ○ Losing GPS signal could cause navigation problems

● Alternatives
  ○ Google Maps
  ○ Internet Search
  ○ Research places in advance
Project Management Plan
Team Organization

● No leader
● Individual roles
  ○ Evan - Planning meetings
  ○ Ryan - Taking meeting minutes
● Communication
  ○ Meetings
  ○ Webcourses
  ○ Email
Software Life Cycle

- Agile
Standards

● Indentation
  ○ Allman style

● Naming Conventions
  ○ PascalCase
  ○ camelCase
  ○ ALL_CAPS

● Comments
  ○ Implementation comments as needed
  ○ Documentation comments for each class, field, and method
Standards

● Documents
  ○ Online templates

● Size metric
  ○ Source lines of code (SLOC)
Development Tools

- Latest versions of Java, Android SDK, and Google APIs
- Configuration Management - GitHub
- Operating system not standardized
Quality Assurance & Risk

- Goals: A variety of real-world testing and user feedback

- Few project specific risks
  - Data integrity
  - Usability
Work Packages and Division

- Initially: UI, Functionality, Documentation
- Further division into: UI Design, Class Development, and Testing
- Individuals responsible for tracking their own progress.
Software Requirements Specifications
Functional Requirements

The mobile application shall allow a user to…

- **add** one or multiple destinations.
- **delete** any particular destination(s).
- **reorder** any particular previously entered destination(s).
- **view all** entered destination(s) for a particular day.
- **navigate**, and see the optimal route navigation between successive items in his/her “to do” list.
Interface Requirements

- Google Maps API for navigation directions.

Physical Environment Requirements

- Run Android version 2.3 or later.
- Internet when navigating.
Users and Human Factors Req’s

- Allow any college level user to successfully navigate through the application with ease.

Documentation Requirements

- Online templates
Data Requirements

- Implement a dynamic programming algorithm to identify the optimal route between destinations.

Security Requirements

- There shall be HTTPS encryption of the communication between the system and server.
Quality Assurance Requirements

● The navigation algorithm to identify the optimal route shall complete in under 30 seconds.
● The user interface will always respond to user interactions within 5 seconds.
Test Plan
Test Environment

- Unit Testing
  - Code base
  - GitHub
- Integration Tests
  - Android emulator
Test Cases

- 1 simple location in area
- 5 locations in area
- Too many locations
- Description too long
- Location out of range
High-Level Design
Use Case Diagram

- User
  - Navigate
  - Reorder Destinations
  - Delete Destination
  - Add Destination
  - Query Google Maps API

- Google Maps
  - «include»

- Modify Destination List
  - «include»
  - «include»
  - «include»
  - «extends»
  - «extends»
  - «extends»

- View Destination List
  - «include»
State Diagram

- User Input
- Navigation
- Click Start Navigation
- Close Navigation

add, remove, reorder, or clear
Data Flow Diagram
Design Issues

● Pipes-and-Filter Architecture
● Functional Decomposition Design
  ○ 2 distinct steps (enter list/navigation)
● Technical Difficulties - Testing GPS
● Performance, Reusability, Reliability, Robustness
Detailed Design
Class Diagram

Activity

InputActivity
- QueryList : List<String>
- add(query : String)
- delete(query : String)
- reorder(id : int, loc : int)
- clear()
- runNavigation()

NavigationActivity
- AddressList : List<Address>
- ListOfDirections : List<String>
- currentSection : int
- NavigationActivity(queryList : List<String>)
- drawMap()
- endNavigation()
- highlight(section : int)
- nextSection()
- prevSection()

APIWrapper
+ drawMap(List<Address>)
+ getDirections(start : Address, end : Address) : List<String>
+ queryPlace(query : String, radius : int)
+ getCurrentLoc() : Address
+ getTime(start : Address, end : Address) : double

PathCalculator
- locations : List<List<Address>>
+ PathCalculator(queryList : List<String>)
+ calculate() : List<Address>
- findLocations()
- chooseOptimal() : List<Address>
Input Activity

Record user input, storing it as a list of “query” strings.

Users can:

● Add items
● Delete items
● Reorder items
● Clear all items

Pressing submit sends the list of queries to the Navigation Activity.
Navigation Activity

- Provides directions to locations
- Updates based on user’s status
- User can view individual sections of the trip
- Gives option to return to input mode

```
NavigationActivity
- AddressList : List<Address>
- ListOfDirections : List<String>
- currentSection : int
- NavigationActivity(queryList : List<String>)
  - drawMap()
  - endNavigation()
  - highlight(section : int)
  - nextSection()
  - prevSection()
```
The PathCalculator class is given a list of queries. It calculates the shortest path to visit places that match the queries in order.
API Wrapper

- Contains wrapper functions for Google’s API
- Uses Maps and Places API

```
APIWrapper
+
  + drawMap(List<Address>)
  + getDirections(start: Address, end: Address): List<String>
  + queryPlace(query: String, radius: int)
  + getCurrentLoc(): Address
  + getTime(start: Address, end: Address): double
```
Requirements Trace

InputActivity class:
- add one or multiple destinations.
- delete any particular destination(s).
- reorder any particular previously entered destination(s).
- view all entered destination(s) for a particular day.

NavigationActivity class
- navigate, and see the optimal route navigation between successive items in his/her “to do” list.

InputActivity and NavigationActivity classes:
- The user interface will always respond to user interactions within 5 seconds
Requirements Trace

PathCalculator class:
- Implement a dynamic programming algorithm to identify the optimal route between destinations.
- The navigation algorithm to identify the optimal route shall complete in under 30 seconds.

APIWrapper class:
- utilize Google Maps API for navigation
- HTTPS encryption of communication between system and server
Questions?