MINISTRY OF EDUCATION AND TRAINING

FPT UNIVERSITY

Capstone Project Document

Parking Guidance System Solution

Group 1	
Group members	Trần Nguyễn Minh Trung – Team Leader – SE61496 Bùi Phú Hiệp – Team Member – SE61438 Nguyễn Đỗ Phương Huy – Team Member – SE61358
Supervisor	Nguyễn Đức Lợi
Ext. Supervisor	N/A
Capstone Project Code	PGSS

- Ho Chi Minh City, Jan, 2017

This page is intentionally left blank

Table of Contents

Table of Contents 1
List of Tables 11
List of Figures 20
Definitions, Acronyms and Abbreviations 25
A. Introduction 26
1. Project Information 26
2. Introduction 26
3. Current Situation 26
3.1. Indoor parking area 27
3.2. Outdoor parking area 28
3.3. Traditional PGS 29
4. Problem Definition 30
5. Proposed Solution 31
5.1. Feature functions 31
5.1.1. Parking Guidance System 31
5.1.2. Mobile app 31
5.2. Advantages 31
5.3. Disadvantages 31
6. Functional Requirements 32
7. Roles and Responsibilities 32
8. Conclusion 33
B. Software – Hardware Project Management Plan 34
1. Problem Definition 34
1.1. Name of this Capstone Project 34
1.2. Problem Abstract 34
1.3. Project Overview 34
1.3.1. Current Situation 34
1.3.2. The Proposed System 34
1.3.2.1. Interaction Block 35
1.3.2.2. Information Block 35

1.3.2.3. Central Control Unit....35
1.3.2.4. Web API Server....35
1.3.2.5. Mobile Application....35
1.3.3. Boundaries of the System....35
1.3.4. Future Plans....36
1.3.5. Development Environment....37
1.3.5.1. Hardware requirements....37
1.3.5.2. Software requirements....38
2. Project organization....38
2.1. Software Process Model....38
2.2. Roles and responsibilities....39
2.3. Tools and Techniques....40
3. Project Management Plan....41
3.1. System development life cycle....41
3.2. Plan Detail....43
3.3. All Meeting Minutes....46
4. Coding Convention....46
4.1. C/C++ Convention....46
4.2. C#, ASP.NET Convention....46
4.3. Python Convention....46
4.4. Android Convention....47
C. Software – Hardware Requirement Specification....48
1. Software Requirement Specification....48
1.1. Software Requirement....48
1.2. GUI Requirement....48
2. Hardware Requirement Specification....48
2.1. Hardware Requirement....48
2.1.1. Hardware Interface....48
2.1.1.1. Block Diagram....49
2.1.1.2. Raspberry Pi 3....50
2.1.1.3. Arduino Nano....51
2.1.1.4. Compass Module 3-Axis HMC5883L....52

2.1.1.5. RF module nRF24L01+ 54
2.1.1.6. Information LED Display Module 55
2.1.1.7. Indicator LED Module 59
2.1.1.8. Servo Motor SG90 63
2.1.2. Communication Protocol 64
2.2. System Overview Use Case 64
2.3. List of Use Case 65
2.3.1. Manager Use Case 65
2.3.2. Administrator Use Case 69
2.3.3. End User Use Case 78
3. Software System Attribute 82
3.1. Usability 82
3.2. Reliability 82
3.3. Availability 82
3.4. Security 82
3.5. Maintainability 82
3.6. Portability 83
3.7. Performance 83
4. Conceptual Diagram 83
D. Software – Hardware Design Description 84
1. Design Overview 84
2. System Architectural Design 85
2.1. Hardware Program Architecture Description 86
2.2. Mobile Application Architecture Description 88
2.3. Web API Architecture Description 89
3. Component Diagram 90
3.1. General Component Diagram 90
3.2. Lot Device Diagram 91
3.3. Information Device Diagram 91
3.4. Server Diagram 92
3.5. Component Dictionary 92
4. Detailed Description 93

4.1. Web API Detailed Description 93
4.1.1. Class Diagram 93
4.1.2. Class Diagram Explanation 98
4.1.2.1. Model Class Diagram 98
4.1.2.2. View Model Class Diagram 100
4.1.2.3. Repository Class Diagram 104
4.1.2.4. Service Class Diagram 105
4.1.3. Interaction Diagram 107
4.1.3.1. Register 108
4.1.3.2. Login 109
4.1.3.3. Get Car Park 109
4.1.3.4. Get Area 110
4.1.3.5. Get Parking Lot 110
4.1.3.6. Update Area 111
4.1.3.7. Update Parking Lot 111
4.1.3.8. Update List Status Parking Lot 112
4.2. Mobile Detailed Description 112
4.2.1. Class Diagram 112
4.2.2. Class Diagram Explanation 123
4.2.2.1. [Activities] MainActivity 123
4.2.2.2. [Activities] CarParkDetailActivity 123
4.2.2.3. [Activities] ManagerActivity 124
4.2.2.4. [Activities] AreaListActivity 125
4.2.2.5. [Activities] ParkingLotListActivity 126
4.2.2.6. [Activities] LoginActivity 127
4.2.2.7. [Activities] RegisterActivity 127
4.2.2.8. [Activities] UserActivity 128
4.2.2.9. [Activities] CarParkListActivity 130
4.2.2.10. [Activities] TransactionActivity 130
4.2.2.11. [Adapters] AreaAdapter 131
4.2.2.12. [Adapters] AreaAdapter.OnItemClickListener 131
4.2.2.13. [Adapters] AreaAdapter.ViewHolder 132

4.2.2.14. [Adapters] CarParkAdapter 132
4.2.2.15. [Adapters] CarParkAdapter.OnItemClickListener 132
4.2.2.16. [Adapters] CarParkAdapter.ViewHolder 133
4.2.2.17. [Adapters] ParkingLotAdapter 133
4.2.2.18. [Adapters] ParkingLotAdapter.OnItemClickListener 133
4.2.2.19. [Adapters] ParkingLotAdapter.ViewHolder 134
4.2.2.20. [Adapters] CarParkAdvanceAdapter 134
4.2.2.21. [Adapters] CarParkAdvanceAdapter.OnItemClickListener 135
4.2.2.22. [Adapters] CarParkAdvanceAdapter.ViewHolder 135
4.2.2.23. [Adapters] TransactionAdapter 135
4.2.2.24. [Adapters] TransactionAdapter.OnItemClickListener 136
4.2.2.25. [Adapters] TransactionAdapter.ViewHolder 136
4.2.2.26. [Adapters] UserInfoWindowAdapter 136
4.2.2.27. [Helpers] AppDatabaseHelper 137
4.2.2.28. [Helpers] PubnubHelper 138
4.2.2.29. [Helpers] AccountHelper 138
4.2.2.30. [Helpers] MapMarkerHelper 139
4.2.2.31. [Helpers] InternetHelper 139
4.2.2.32. [Interfaces] AreaClient 139
4.2.2.33. [Interfaces] TransactionClient 139
4.2.2.34. [Interfaces] CarParkClient 140
4.2.2.35. [Interfaces] AccountClient 140
4.2.2.36. [Interfaces] ParkingLotClient 140
4.2.2.37. [Models] Account 141
4.2.2.38. [Models] Area 141
4.2.2.39. [Models] CheckCode 142
4.2.2.40. [Models] Geo 142
4.2.2.41. [Models] CarPark 142
4.2.2.42. [Models] Transaction 143
4.2.2.43. [Models] ParkingLot 144
4.2.2.44. [Models] TransactionStatus 144
4.2.2.45. [Models] AreaStatus 144

4.2.2.46. [Models] ParkingLotStatus 145
4.2.2.47. [Network] AccountPackage 145
4.2.2.48. [Network] AreaPackage 146
4.2.2.49. [Network] CarParkPackage 146
4.2.2.50. [Network] CheckCodePackage 146
4.2.2.51. [Network] CommandPackage 147
4.2.2.52. [Network] RealtimeMapPackage 147
4.2.2.53. [Network] ControlPubnubPackage 147
4.2.2.54. [Network] MobilePubnubPackage 148
4.2.2.55. [Network] GetCoordinationPackage 148
4.2.2.56. [Network] CarParkAdvancePackage 148
4.2.2.57. [Network] NotificationPackage 149
4.2.2.58. [Network] TransactionPackage 149
4.2.2.59. [Network] ParkingLotPackage 150
4.2.2.60. [Network] ServiceGenerator 150
4.2.2.61. [Network] DateTypeDeserializer 150
4.2.3. Interaction Diagram 151
4.2.3.1. Load car parks on map 151
4.2.3.2. Load car parks in list 152
4.2.3.3. Car park detail information 152
4.2.3.4. Reserve parking lot process 153
4.2.3.5. Transaction controller 154
4.2.3.6. Manage car parks 156
4.2.3.7. Manage areas 157
4.2.3.8. Manage parking lots 158
4.2.3.9. Check reservation PIN code 159
4.3. Embedded Program Detailed Description 160
4.3.1. Class Diagram 160
4.3.2. Class Diagram Explanation 163
4.3.2.1. [Lot] Main 163
4.3.2.2. [Lot] HMC5883L 164
4.3.2.3. [Lot] RGBLED 164

4.3.2.4. [Lot] common_type 165
4.3.2.5. [Sign] Main 165
4.3.2.6. [Sign] SEVEN_SEGMENT 166
4.3.2.7. [Lot/Sign] RFUtil 166
4.3.2.8. [Lot/Sign] CRC24 167
4.3.2.9. [Hub] hub 168
4.3.2.10. [Hub] rf_meta 169
4.3.2.11. [Hub] crc24 170
4.3.2.12. [Hub] api_service 170
4.3.2.13. [Hub] pubnub_meta 170
4.3.2.14. [Hub] Area 171
4.3.2.15. [Hub] ParkingLot 171
4.3.2.16. [Hub] UserPackage 172
4.3.3. Interaction Diagram 172
4.3.3.1. Lot node process 172
4.3.3.2. Sign node process 174
4.3.3.3. Hub polling process 175
4.3.3.1. Hub execute request process 177
4.4. Hardware Detailed Description 179
4.4.1. Raspberry Pi 3 model B 179
4.4.2. Arduino Nano 181
4.4.3. Compass module 3-Axis HMC5883L 183
4.4.4. RF module nRF24L01+ 186
4.4.5. Lot node 191
4.4.5.1. RGB LED common anode 191
4.4.5.2. Transistor TIP122 193
4.4.5.3. Servo motor SG90 194
4.4.5.4. LM7805 196
4.4.6. Sign node 197
4.4.6.1. Power logic 8-bit shift register TPIC6B595 197
4.4.6.2. 7-Segment LED display 200
4.4.7. Components connection 201

4.4.7.1. Overview component connection 201
4.4.7.2. Raspberry Pi 3 connection 203
4.4.7.3. Lot node connection 203
4.4.7.4. Sign node connection 203
5. Interface 204
5.1. Component Interface 204
5.2. User Interface Design 206
5.2.1. Common Interface 206
5.2.1.1. Login screen 206
5.2.1.2. Register screen 207
5.2.2. User/Guest Interface 209
5.2.2.1. Map view screen 209
5.2.2.2. Navigation view 211
5.2.2.3. Car park list view screen 213
5.2.2.4. Car park detail view screen 214
5.2.2.5. History transaction list view screen 216
5.2.2.6. History transaction detail view screen 217
5.2.2.7. Reservation process view screen 219
5.2.3. Manager Interface 222
5.2.3.1. Car park list view screen 222
5.2.3.2. Car park detail view screen 224
5.2.3.3. Area list view screen 225
5.2.3.4. Area detail view screen 226
5.2.3.5. Parking lot list view screen 227
5.2.3.6. Parking lot detail view screen 229
5.2.3.7. Check code view screen 230
6. Database Design 232
6.1. Logical Diagram 232
6.2. Data Dictionary 232
7. Algorithms 234
7.1. GetDistance formula 234
7.2. CRC Error Detection 234

7.2.1. Definition 234
7.2.2. Define Problem 234
7.2.3. Solution theory 234
7.2.4. Implementation 237
7.2.5. Table-driven implementation 238
E. System Implementation & Test 242
1. Introduction 242
1.1. Overview 242
1.2. Test Approach 242
2. Database Relationship Diagram 242
2.1. Physical Diagram 242
2.2. Data Dictionary 243
3. Performance Measures 246
4. Test Plan 247
4.1. Features to be tested 247
4.1.1. Hardware 248
4.1.2. Software 249
4.2. Features not to be tested: 249
4.3. Test environment 249
4.4. Test Pass/Fail Criteria 249
5. System Testing Test Case 250
5.1. Component Testing 251
5.2. API Web Service Testing 253
5.3. Mobile Testing 255
F. User's Manual 258
1. Installation Guide 258
1.1. Hardware installation 258
1.2. Hub installation 258
1.3. Mobile Application 259
2. User Guide 260
2.1. Manager 260
2.1.1. Edit car park information 260

2.1.2. Edit area information 262
2.1.3. Edit parking lot information 264
2.1.4. Check PIN code for user 266
2.2. User 268
2.2.1. Search for car park near user in map 268
2.2.2. Search for car park near location in map 270
2.2.3. Search for car park near user/location in list view 272
2.2.4. Call the car park 274
2.2.5. Find best route from current location to car park 276
2.2.6. Reserve a parking lot in car park 278
2.2.7. Cancel/Check-in reservation 281
G. Appendix 284

List of Tables

Table 1: Definitions, Acronyms and Abbreviations 25
Table 2: Roles and Responsibilities 32
Table 3: Database requirement 37
Table 4: API Service Requirement 37
Table 5: Provide CCU Hardware 37
Table 6: Roles and Responsibilities Details 40
Table 7: Tools 40
Table 8: Techniques 41
Table 9: System Development Life Cycle 42
Table 10: System Development Detail Plan 45
Table 11: Raspberry Pi 3 – Specification 50
Table 12: Arduino Nano - Specification 52
Table 13: The Compass Module 3-Axis HMC5883L - Pin Function 53
Table 14: RF Module nRF24L01 – Pin function 55
Table 15: IC TPIC6B595 - Pin Function 59
Table 16: Servo Motor SG90 – Pin-outs 64
Table 17: Conceptual diagram data dictionary 83
Table 18: Component dictionary 93
Table 19: CarPark class attributes 98
Table 20: CarParkWithAmountEntities class attributes 98
Table 21: Area class attributes 98
Table 22: AreaCustom class attributes 99
Table 23: ParkingLot class attributes 99
Table 24: Transaction class attributes 99
Table 25: AspNetUser class attributes 100
Table 26: CarParkViewModel class attributes 100
Table 27: CarParkWithAmount class attributes 101
Table 28: CarParkUpdateViewModel class attributes 101
Table 29: AreaViewModel class attributes 101
Table 30: AreaCustomViewModel class attributes 101

Table 31: ParkingLotViewModel class attributes 102
Table 32: ParkingLotUpdateViewModel class attributes 102
Table 33: Transaction class attributes 102
Table 34: TransactionCustomViewModel class attributes 103
Table 35: TransactionCreateViewModel class attributes 103
Table 36: TransactionCreateReturnViewModel class attributes 103
Table 37: TransactionUpdateViewModel class attributes 103
Table 38: AspNetUser class attributes 104
Table 39: BaseRepository class methods 105
Table 40: BaseService class methods 106
Table 41: AreaService class methods 106
Table 42: CarParkService class methods 106
Table 43: ParkingLotService class methods 107
Table 44: TransactionService class methods 107
Table 45: Packages dictionary 112
Table 46: Activities package class dictionary 114
Table 47: Adapters package class dictionary 117
Table 48: Helpers package class dictionary 117
Table 49: Interfaces package class dictionary 118
Table 50: Models package class dictionary 120
Table 51: Networks package class dictionary 122
Table 52: [Activities] MainActivity methods 123
Table 53: [Activities] CarParkDetailActivity attributes 123
Table 54: [Activities] CarParkDetailActivity methods 124
Table 55: [Activities] ManagerActivity attributes 125
Table 56: [Activities] ManagerActivity methods 125
Table 57: [Activities] AreaListActivity attributes 126
Table 58: [Activities] AreaListActivity methods 126
Table 59: [Activities] ParkingLotListActivity attributes 127
Table 60: [Activities] ParkingLotListActivity methods 127
Table 61: [Activities] LoginActivity attributes 127
Table 62: [Activities] LoginActivity methods 127

Table 63: [Activities] RegisterActivity attributes 128
Table 64: [Activities] RegisterActivity methods 128
Table 65: [Activities] UserActivity attributes 129
Table 66: [Activities] UserActivity methods 129
Table 67: [Activities] CarParkListActivity attributes 130
Table 68: [Activities] CarParkListActivity methods 130
Table 69: [Activities] TransactionActivity attributes 130
Table 70: [Activities] TransactionActivity methods 131
Table 71: [Adapters] AreaAdapter attributes 131
Table 72: [Adapters] AreaAdapter methods 131
Table 73: [Adapters] AreaAdapter.OnItemClickListener methods 131
Table 74: [Adapters] AreaAdapter.ViewHolder attributes 132
Table 75: [Adapters] AreaAdapter.ViewHolder methods 132
Table 76: [Adapters] CarParkAdapter attributes 132
Table 77: [Adapters] CarParkAdapter methods 132
Table 78: [Adapter] CarParkAdapter.OnItemClickListener methods 133
Table 79: [Adapters] CarParkAdapter.ViewHolder attributes 133
Table 80: [Adapters] CarParkAdapter.ViewHolder methods 133
Table 81: [Adapters] ParkingLotAdapter attributes 133
Table 82: [Adapters] ParkingLotAdapter methods 133
Table 83: [Adapters] ParkingLotAdapter.OnItemClickListener methods 134
Table 84: [Adapters] ParkingLotAdapter.ViewHolder attributes 134
Table 85: [Adapters] ParkingLotAdapter.ViewHolder methods 134
Table 86: [Adapters] CarParkAdvanceAdapter attributes 134
Table 87: [Adapters] CarParkAdvanceAdapter methods 134
Table 88: [Adapters] CarParkAdvanceAdapter.OnItemClickListener methods 135
Table 89: [Adapters] CarParkAdvanceAdapter.ViewHolder attributes 135
Table 90: [Adapters] CarParkAdvanceAdapter.ViewHolder methods 135
Table 91: [Adapters] TransactionAdapter attributes 135
Table 92: [Adapters] TransactionAdapter methods 136
Table 93: [Adapters] TransactionAdapter.OnItemClickListener methods 136
Table 94: [Adapters] TransactionAdapter.ViewHolder attributes 136

Table 95: [Adapters] TransactionAdapter.ViewHolder methods 136
Table 96: [Adapters] UserInfoWindowAdapter attributes 137
Table 97: [Adapters] UserInfoWindowAdapter methods 137
Table 98: [Helpers] AppDatabaseHelper attributes 137
Table 99: [Helpers] AppDatabaseHelper methods 138
Table 100: [Helpers] PubnubHelper attributes 138
Table 101: [Helpers] PubnubHelper methods 138
Table 102: [Helpers] AccountHelper attributes 138
Table 103: [Helpers] AccountHelper methods 139
Table 104: [Helpers] MapMarkerHelper methods 139
Table 105: [Helpers] InternetHelper methods 139
Table 106: [Interfaces] AreaClient methods 139
Table 107: [Interfaces] TransactionClient methods 140
Table 108: [Interfaces] CarParkClient methods 140
Table 109: [Interfaces] AccountClient methods 140
Table 110: [Interfaces] ParkingLotClient methods 141
Table 111: [Models] Account attributes 141
Table 112: [Models] Account methods 141
Table 113: [Models] Area attributes 141
Table 114: [Models] Area methods 141
Table 115: [Models] CheckCode attributes 142
Table 116: [Models] CheckCode methods 142
Table 117: [Models] Geo attributes 142
Table 118: [Models] Geo methods 142
Table 119: [Models] CarPark attributes 143
Table 120: [Models] CarPark methods 143
Table 121: [Models] Transaction attributes 143
Table 122: [Models] Transaction methods 143
Table 123: [Models] ParkingLot attributes 144
Table 124: [Models] ParkingLot methods 144
Table 125: [Models] TransactionStatus attributes 144
Table 126: [Models] TransactionStatus methods 144

Table 127: [Models] AreaStatus attributes 145
Table 128: [Models] AreaStatus methods 145
Table 129: [Models] ParkingLotStatus attributes 145
Table 130: [Models] ParkingLotStatus methods 145
Table 131: [Network] AccountPackage attributes 145
Table 132: [Network] AccountPackage methods 146
Table 133: [Network] AreaPackage attributes 146
Table 134: [Network] AreaPackage methods 146
Table 135: [Network] CarParkPackage attributes 146
Table 136: [Network] CarParkPackage methods 146
Table 137: [Network] CheckCodePackage attributes 146
Table 138: [Network] CheckCodePackage methods 147
Table 139: [Network] CommandPackage attributes 147
Table 140: [Network] CommandPackage methods 147
Table 141: [Network] RealtimeMapPackage attributes 147
Table 142: [Network] RealtimeMapPackage methods 147
Table 143: [Network] ControlPubnubPackage attributes 148
Table 144: [Network] ControlPubnubPackage methods 148
Table 145: [Network] MobilePubnubPackage attributes 148
Table 146: [Network] MobilePubnubPackage methods 148
Table 147: [Network] GetCoordinationPackage attributes 148
Table 148: [Network] GetCoordinationPackage methods 148
Table 149: [Network] CarParkAdvancePackage attributes 149
Table 150: [Network] CarParkAdvancePackage methods 149
Table 151: [Network] NotificationPackage attributes 149
Table 152: [Network] NotificationPackage methods 149
Table 153: [Network] TransactionPackage attributes 149
Table 154: [Network] TransactionPackage methods 149
Table 155: [Network] ParkingLotPackage attributes 150
Table 156: [Network] ParkingLotPackage methods 150
Table 157: [Network] ServiceGenerator attributes 150
Table 158: [Network] ServiceGenerator methods 150

Table 159: [Network] DateTypeDeserializer attributes 150
Table 160: [Network] DateTypeDeserializer methods 151
Table 161: Lot node class dictionary 160
Table 162: Sign node class dictionary 161
Table 163: Hub class dictionary 163
Table 164: [Lot] Main attributes 163
Table 165: [Lot] Main methods 163
Table 166: [Lot] HMC5883L attributes 164
Table 167: [Lot] HMC5883L methods 164
Table 168: [Lot] RGBLED attributes 164
Table 169: [Lot] RGBLED methods 165
Table 170: [Lot] common_type attributes 165
Table 171: [Sign] Main attributes 165
Table 172: [Sign] Main methods 166
Table 173: [Sign] SEVEN_SEGMENT attributes 166
Table 174: [Sign] SEVEN_SEGMENT methods 166
Table 175: [Lot/Sign] RFUtil attributes 167
Table 176: [Lot/Sign] RFUtil methods 167
Table 177:: [Lot/Sign] CRC24 attributes 167
Table 178: [Lot/Sign] CRC24 methods 167
Table 179: [Hub] hub attributes 168
Table 180: [Hub] hub methods 169
Table 181: [Hub] rf_meta attributes 169
Table 182: [Hub] rf_meta methods 170
Table 183: [Hub] crc24 attributes 170
Table 184: [Hub] crc24 methods 170
Table 185: [Hub] api_service methods 170
Table 186: [Hub] pubnub_meta attributes 171
Table 187: [Hub] pubnub_meta methods 171
Table 188: [Hub] Area attributes 171
Table 189: [Hub] ParkingLot attributes 172
Table 190: [Hub] UserPackage attributes 172

Table 191: HMC55883L Register list 185
Table 192: nRF24L01+ pinout 187
Table 193: nRF24L01+ SPI commands 191
Table 194: Servo motor SG90 pinout 194
Table 195: Power logic 8-bit shift register TPIC6B595 pinout 199
Table 196: Raspberry Pi 3 connection 203
Table 197: Lot node connection 203
Table 198: Sign node connection 203
Table 199: TPIC6B595s sequence connection 204
Table 200: Login screen fields 206
Table 201: Login screen button/hyperlinks 207
Table 202: Register screen fields 208
Table 203: Register screen button/hyperlinks 208
Table 204: Map view screen fields 210
Table 205: Map view screen button/hyperlinks 210
Table 206: Navigation view fields 211
Table 207: Navigation view button/hyperlinks 212
Table 208: Car park list view screen fields 214
Table 209: Car park list view screen button/hyperlinks 214
Table 210: Car park detail view screen fields 215
Table 211: Car park detail view screen button/hyperlinks 215
Table 212: History transaction list view screen fields 217
Table 213: History transaction list view screen button/hyperlinks 217
Table 214: History transaction detail view screen fields 218
Table 215: History transaction detail view screen button/hyperlinks 218
Table 216: Reservation process view screen 1 fields 219
Table 217: Reservation process view screen 1 button/hyperlinks 220
Table 218: Reservation process view screen 2 fields 220
Table 219: Reservation process view screen 2 button/hyperlinks 221
Table 220: Reservation process notification fields 222
Table 221: Car park list view screen fields 223
Table 222: Login screen button/hyperlinks 223

Table 223: Car park detail view screen fields 224
Table 224: Car park detail view screen button/hyperlinks 225
Table 225: Area list view screen fields 226
Table 226: Area list view screen button/hyperlinks 226
Table 227: Area detail view screen fields 227
Table 228: Area detail view screen button/hyperlinks 227
Table 229: Parking lot list view screen fields 228
Table 230: Parking lot list view screen button/hyperlinks 228
Table 231: Parking lot detail view screen fields 229
Table 232: Parking lot detail view screen button/hyperlinks 230
Table 233: Check code screen fields 231
Table 234: Check code screen button/hyperlinks 231
Table 235: Database data dictionary 233
Table 236: Data dictionary 243
Table 237: Data dictionary detail 246
Table 238: Hardware devices test list 249
Table 239: Software test list 249
Table 240: Component test 252
Table 241: API Web service test 255
Table 242: Mobile test 257
Table 243: Edit car park information step 1 260
Table 244: Edit car park information step 2 261
Table 245: Edit area information step 1 262
Table 246: Edit area information step 2 263
Table 247: Edit parking lot information step 1 264
Table 248: Edit parking lot information step 2 265
Table 249: Check PIN code for user step 1, 2 266
Table 250: Check PIN code for user step 3 267
Table 251: Search for car park near user in map step 1, 2 268
Table 252: Search for car park near user in map step 3, 4 269
Table 253: Search for car park near location in map step 1, 2, 3 270
Table 254: Search for car park near location in map step 4, 5, 6 271

Table 255: Search for car park near user/location in list view step 1, 2 272
Table 256: Search for car park near user/location in list view step 3 273
Table 257: Call the car park step 1 274
Table 258: Call the car park step 2 275
Table 259: Find best route from current location to car park step 1 276
Table 260: Find best route from current location to car park step 2 277
Table 261: Reserve a parking lot in car park step 1 278
Table 262: Reserve a car park in parking lot step 2 279
Table 263: Reserve a parking lot in car park step 3 280
Table 264: Cancel/Check-in reservation step 1, 2 281
Table 265: Cancel/Check-in reservation step 3 282
Table 266: Cancel/Check-in reservation step 4 283

List of Figures

Figure 1: Indoor parking area 27
Figure 2: Outdoor parking area 28
Figure 3: Parking area with PGS 29
Figure 4: Zone Control Unit 30
Figure 5: Project Block Diagram 36
Figure 6: Iterative and Incremental development 39
Figure 7: PGSS Block Diagram 49
Figure 8: Raspberry Pi 3 50
Figure 9: Arduino Nano 51
Figure 10: Compass Module 3-Axis HMC5883L 52
Figure 11: RF module nRF24L01+ 54
Figure 12: RF module nRF24L01+ - Specification 55
Figure 13: 7-segment LED Display 56
Figure 14: TPIC6B595 Power Logic 8-Bit Shift Register 57
Figure 15: TPIC6B595 Pin-outs 58
Figure 16: RGB LED common anode 59
Figure 17: RGB LED common anode pin-out 60
Figure 18: Overview Use Case Diagram 64
Figure 19: Manager Use Case Diagram 65
Figure 20: Manager Use Case Diagram 69
Figure 21: End User Use Case Diagram 78
Figure 22: Conceptual Diagram 83
Figure 23: System architecture design 85
Figure 24: Hardware Program architectural 86
Figure 25: Mobile Application architecture 88
Figure 26: Hardware - Software Connection Architecture Description 89
Figure 27: General Component Diagram 90
Figure 28: Lot Device Diagram 91
Figure 29: Information Device Diagram 91
Figure 30: Server Diagram 92

Figure 31: Model Class Diagram 94
Figure 32: View Model Class Diagram 95
Figure 33: Repository Class Diagram 96
Figure 34: Service Class Diagram 97
Figure 35: Register Diagram 108
Figure 36: Login Diagram 109
Figure 37: GetCarPark Diagram 109
Figure 38: GetArea Diagram 110
Figure 39: GetParkingLot Diagram 110
Figure 40: UpdateArea Diagram 111
Figure 41: UpdateParkingLot Diagram 111
Figure 42: UpdateListStatusParkingLot Diagram 112
Figure 43: Main class diagram and package view 112
Figure 44: Activities package class diagram 113
Figure 45: Adapters package class diagram 115
Figure 46: Helpers package class diagram 117
Figure 47: Interfaces package class diagram 118
Figure 48: Models package class diagram 119
Figure 49: Networks package class diagram 121
Figure 50: Interaction diagram - Load car parks on map 151
Figure 51: Interaction diagram - Load car parks in list 152
Figure 52: Interaction diagram - Car park detail information 153
Figure 53: Interaction diagram - Reserve parking lot process 154
Figure 54: Interaction diagram - Transaction controller 155
Figure 55: Interaction diagram - Manage car parks 156
Figure 56: Interaction diagram - Manage areas 157
Figure 57: Interaction diagram - Manage parking lots 158
Figure 58: Interaction diagram - Check reservation PIN code 159
Figure 59: Lot node class diagram 160
Figure 60: Sign node class diagram 161
Figure 61: Hub class diagram 162
Figure 62: Interaction diagram - Lot node process 173

Figure 63: Interaction diagram - Sign node 174
Figure 64: Interaction diagram - Hub polling process 176
Figure 65: Interaction diagram - Hub request process 178
Figure 66: Raspberry Pi 3 model B 179
Figure 67: Raspberry Pi 3 Model B pinount 180
Figure 68: Arduino Nano 181
Figure 69: Arduino Nano pinout 182
Figure 70: 3-Axis HMC5883L 183
Figure 71: Earth's magnetic field lines distorted by ferrous metal of the car 184
Figure 72: Arduino Nano and HMC5883L connection 184
Figure 73: RF module nRF24L01+ 186
Figure 74: nRF24L01+ pinout 187
Figure 75: Arduino Nano and nRF24L01+ connection 188
Figure 76: nRF24L01+ packet format 189
Figure 77: RGB LED common anode 191
Figure 78: Arduino Nano and RGB LED connection through TIP122 192
Figure 79: Transistor TIP122 193
Figure 80: Transistor TIP122 pinout 193
Figure 81: Servo motor SG90 194
Figure 82: Arduino Nano and Servo SG90 connection, with LM7805 195
Figure 83: LM7805 196
Figure 84: Power logic 8-bit shift register TPIC6B595 197
Figure 85: Power logic 8-bit shift register TPIC6B595 pinout 198
Figure 86: Arduino Nano and TPIC6B595 connection 199
Figure 87: Sequence connection between 2 TPIC6B595 200
Figure 88: 7-Segment LED display 200
Figure 89: General component connection 201
Figure 90: Lot node connection 202
Figure 91: Sign node connection 202
Figure 92: Lot node interface 204
Figure 93: Sign node interface 205
Figure 94: Login screen 206

Figure 95: Register screen 207
Figure 96: Map view screen 209
Figure 97: Navigation view 211
Figure 98: Car park list view screen 213
Figure 99: Car park detail view screen 214
Figure 100: History transaction list view screen 216
Figure 101: History transaction detail view screen 217
Figure 102: Reservation process view screen 1 219
Figure 103: Reservation process view screen 2 220
Figure 104: Reservation process notification 221
Figure 105: Car park list view screen 222
Figure 106: Car park detail view screen 224
Figure 107: Area list view screen 225
Figure 108: Area detail view screen 226
Figure 109: Parking lot list view screen 227
Figure 110: Parking lot detail view screen 229
Figure 111: Check code view screen 230
Figure 112: Database logical diagram 232
Figure 113: CRC arithmetic addition 235
Figure 114: CRC arithmetic subtraction 235
Figure 115: CRC arithmetic multiplication 235
Figure 116: CRC arithmetic division 236
Figure 117: CRC implementation register 237
Figure 118: CRC implementation simple flow 238
Figure 119: CRC 32-bits register 239
Figure 120: CRC implementation table-driven flow 240
Figure 121: CRC24 precomputed table 241
Figure 122: Database physical diagram 242
Figure 123: Performance test result 247
Figure 124: Manager, End User Core F 250
Figure 125: Update Raspbian 258
Figure 126: Install and enable GPIO on Raspbian 259

Figure 127: Edit car park information step 1 260
Figure 128: Edit car park information step 2 261
Figure 129: Edit area information step 1 262
Figure 130: Edit area information step 2 263
Figure 131: Edit parking lot information step 1 264
Figure 132: Edit parking lot information step 2 265
Figure 133: Check PIN code for user step 2 266
Figure 134: Check PIN code for user step 3 267
Figure 135: Search for car park near user in map step 1, 2 268
Figure 136: Search for car park near user in map step 3, 4 269
Figure 137: Search for car park near location in map step 1 270
Figure 138: Search for car park near location in map step 4 271
Figure 139: Search for car park near user/location in list view step 2 272
Figure 140: Search for car park near user/location in list view step 3 273
Figure 141: Call the car park step 1 274
Figure 142: Call the car park step 2 275
Figure 143: Find best route from current location to car park step 1 276
Figure 144: Find best route from current location to car park step 2 277
Figure 145: Reserve a parking lot in car park step 1 278
Figure 146: Reserve a car park in parking lot step 2 279
Figure 147: Reserve a parking lot in car park step 3 280
Figure 148: Cancel/Check-in reservation step 2 281
Figure 149: Cancel/Check-in reservation step 3 282
Figure 150: Cancel/Check-in reservation step 4 283

Definitions, Acronyms and Abbreviations

Name	Definition
PGS	Parking Guidance System
Parking area	An area set aside for parking vehicles, aircraft, etc.
Parking lot	A place inside parking area that provide space for one vehicle
IoT	Internet of Things
CCU	Central Control Unit
ASIC	Application-specific integrated circuit
LCC	leadless chip carrier

Table 1: Definitions, Acronyms and Abbreviations

A. Introduction

1. Project Information

- Project name: Parking Guidance System Solution
- Project Code: PGSS
- Product Type: Internet of Things Application
- Start Date: 3-Jan-2017
- End Date: 24-Apr-2017

2. Introduction

Information and guidance system is designed the monitoring and provision of information on the occupancy of individual parking lots in the parking area. The system represents a solution to the current problem of a high proportion of a traffic generated by drivers seeking vacant parking spaces. The guidance system is able to provide drivers with the latest and dynamically changing information on the availability status of monitored parking lots. Using clear guidance signs, vehicles are guided directly to identified vacant parking lots that are the closest to vehicles' current positions.

With the help of the parking guidance system, drivers are able to find a vacant parking lot quickly and easily. The resulting benefits are the reduction of stop-start traffic, pleasant experience of parking, elimination of stressful situations and positive attitude towards the car park operator. The reduction of traffic minimizes the occurrence of traffic accidents. The positive mental state of drivers is important for all commercial subjects that need to stimulate required shopping behavior, repeated visits and the increase of customers' loyalty. In highly competitive environment, the parking guidance system may become a competitive advantage and generate additional profits for operators.

3. Current Situation

The current situation can be summarized through the following 3 categories:

3.1. Indoor parking area

Figure 1: Indoor parking area

Indoor parking areas are growing with the increasing number of vehicles in a developing economy and causing many problems due to multiple floors, followed by zones, distributed parking lots and absence of parking guidance to vehicles drivers. The traditional method of having to navigate around searching for an empty parking lot causes many troubles for drivers, as well as traffic jam in parking areas.

3.2. Outdoor parking area

Figure 2: Outdoor parking area

With the increasing number of vehicles, it creates lots of issues to build a parking building, or a basement plus some other kind of building on top, especially cost and planning structure. There is no other way except to utilize outdoor spaces of places like public parks, mall, hospital… as an outdoor parking areas.

3.3. Traditional PGS

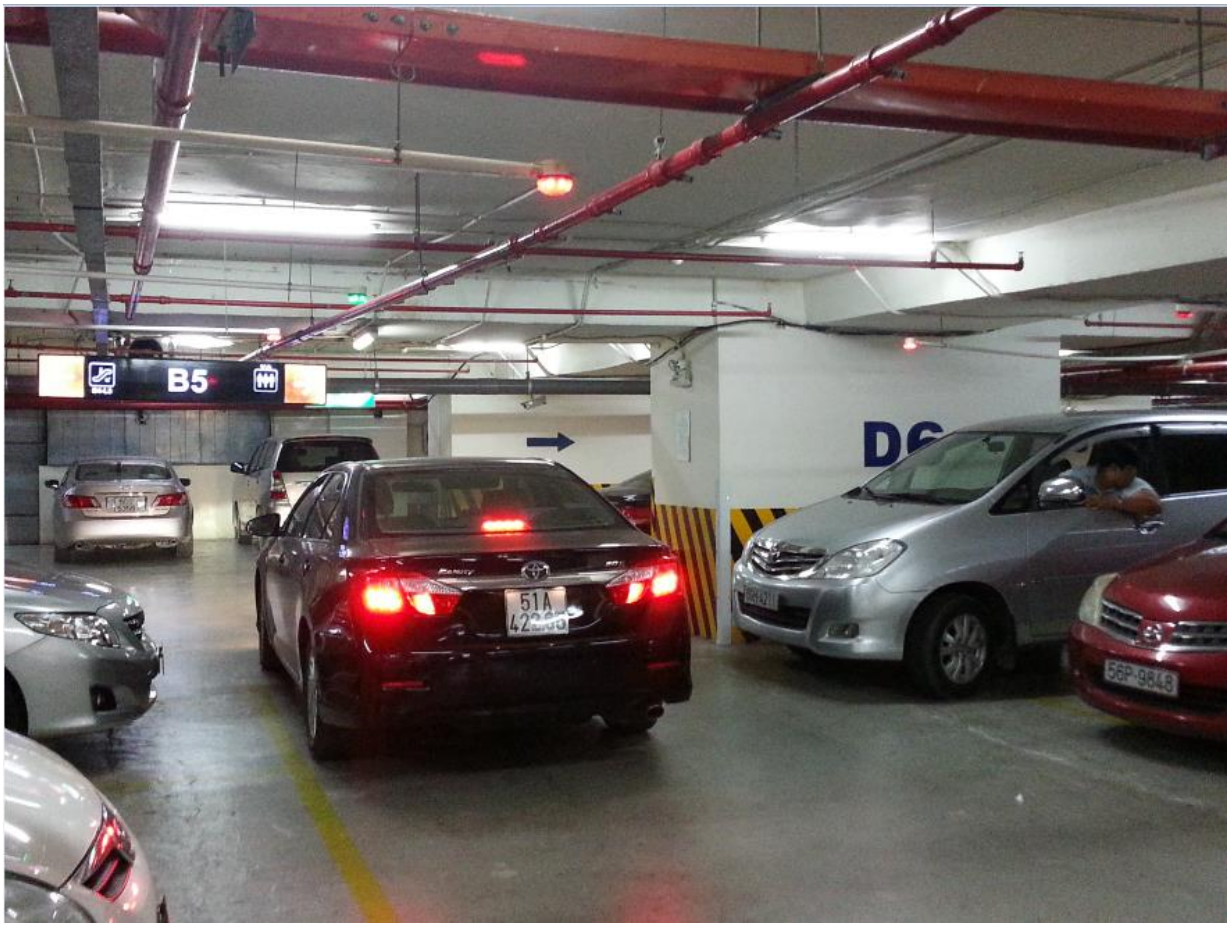


Figure 3: Parking area with PGS

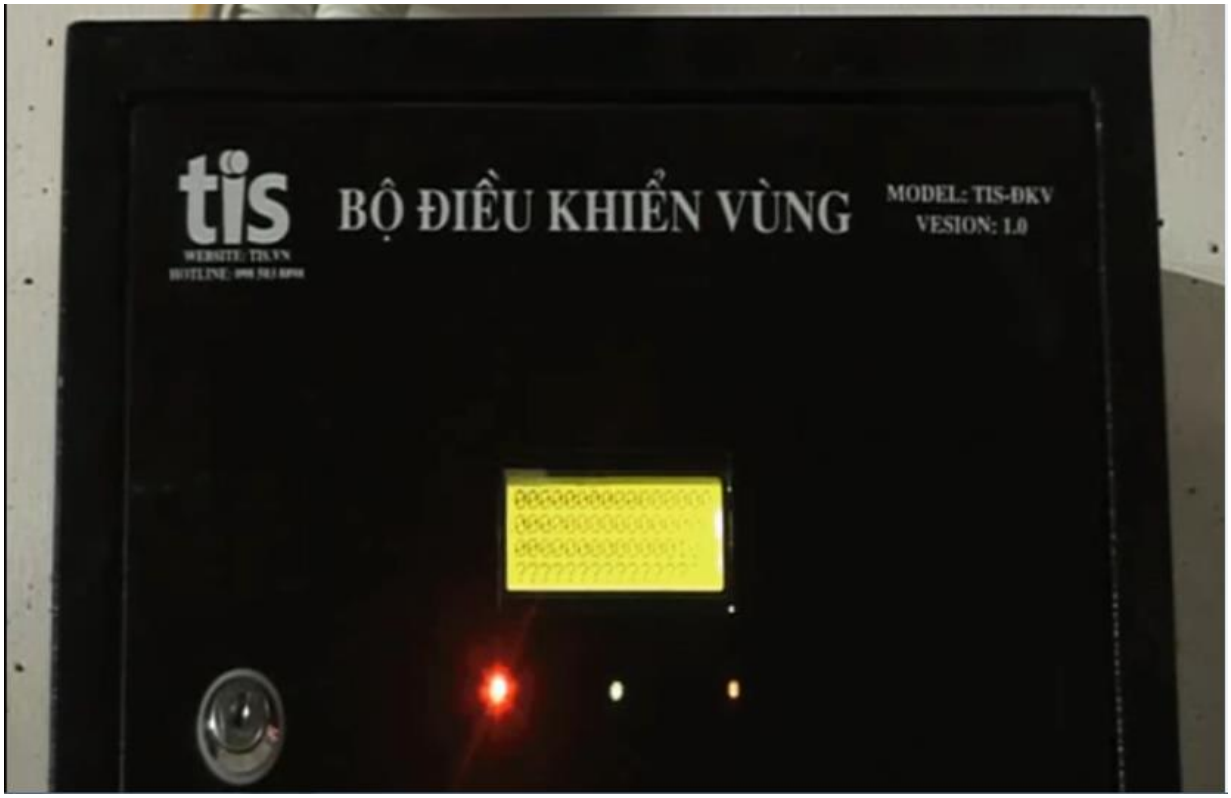


Figure 4: Zone Control Unit

As opposed to the traditional parking areas, the parking areas with PGS keep parking lots under systematic real-time monitoring so as drivers can see what parking spaces are available immediately and with minimal effort. By implementing this, operators also have the chance of increasing their revenues because of the increasing number of satisfied drivers.

The current version of PGS that implemented in a large number parking areas in Vietnam made use of RS485. Each parking lot is fitted with an ultrasonic detector and Indicator light, hence information displays at main entrance and at internal junction points are driven with real-time occupancy detected by ultrasonic detectors. All status sent to a Zone Control Unit on RS485, which in turn be sent to the Central Control Unit.

4. Problem Definition

The current version of PGS is working well but it still has some disadvantages:

- The system implements the RS485 so each Zone Control Unit can have maximum 8 loops of 32 Detectors hence supporting 256 parking lots with 8 Information LED Displays. This is fine for most of the current parking areas, but it provides complicated in a parking area with large number of parking lots like the 6 multi-story car parks with around 7000 parking lots each proposed by Ho Chi Minh City Transport Department.
- The Zone Control Units need to be wired with Detectors, Indicator LED, Information LED Displays and Central Control Unit hence the wiring is pretty

complicated and need careful planning in the construction stage. Therefore, the current version of PGS is hard to implement in most of the existed parking areas.
- The current version of PGS is difficult to use for outdoor parking areas because of the need of installation of the frame.
- Drivers can only get the information of available parking lots at the entrance of parking areas, so the issues of high proportion of traffic generated by drivers seeking vacant parking lots still remains.

5. Proposed Solution

The current version of PGS contains many flaws and proved to be unacceptable for a greater business. Therefore, our proposed solution is to build a parking guidance system with RF modules. The RF modules provide wireless communication directly between Central Control Unit and Detectors, Indicator LED, Information LED Displays so there is no need for the Zone Control Units.

5.1. Feature functions

5.1.1. Parking Guidance System

- Detectors sends out ultrasonic signals from the bottom upward and transmits the signals to the guidance units through RF.
- The Indicator LED, Information LED Displays also use RF communication so they are easier to install.
- The Central Control Unit connect with data stream network to provide real-time information to the app.

5.1.2. Mobile app

- Management portal for operators to setup and manage theirs parking area.
- Customer portal where drivers can view on maps the real time information of nearby parking areas or search for one.

5.2. Advantages

- Fast orientation of drivers when seeking vacant parking lots
- Minimizing the time needed for finding a vacant parking lot
- Improvement of safety, the increase of the traffic effectiveness and efficiency
- Decreases of exhaust fumes as well as the negative impact of traffic on the environment
- Maximum use of the entire car park capacity
- Easy to assemble

5.3. Disadvantages

- System does not provide car find feature
- The detector can only detect at the location above it so it can’t detect if there is anything around the corner of parking lot
- Management portal does not have web version

6. Functional Requirements

Function requirements of the system are listed as below:

- Detector component:
  - Sensor (ultrasonic, infrared, magnetic field, load cell…)
  - RF communication
- Indicator LED component:
  - RGB led controller
  - RF communication
- Information LED Displays component:
  - Main entrance LED Display
  - Internal junction points LED Display
  - RF communication
- Reservation Barrier component:
  - Servo motor controller
  - RF communication
- Central Control System component:
  - Data Stream Network
  - Web API communication
  - RF communication
- Management portal app component:
  - Parking Area Setup
  - Parking Area Analysis
  - Parking Lot Control
- Customer portal app component:
  - Parking Areas Search
  - Parking Lot Reservation
- Web API component:
  - Parking Areas Search
  - Parking Area Setup
  - Parking Area Analysis
  - Parking Lot Status

7. Roles and Responsibilities

No	Full name	Role	Position	Contact
1	Nguyễn Đức Lợi	Project manager	Supervisor	loind@fpt.edu.vn
2	Trần Nguyễn Minh Trung	Developer	Leader	trungtnmse61496@fpt.edu.vn
3	Bùi Phú Hiệp	Developer	Member	hiepbpse61438@fpt.edu.vn
4	Nguyễn Đỗ Phương Huy	Developer	Member	huyndpse61358@fpt.edu.vn

Table 2: Roles and Responsibilities

8. Conclusion

For this project, we will try to to reproduce the traditional Parking Guidance System with wireless technology, add a web server to manage information, a mobile app to provide UI for normal users and parking area operators to make this more like an IoT application. Therefore, we will need to:

- Research to determine and implement the appropriate MCU for the Central Control Unit and other nodes (**Arduino, Raspberry**, CC1310…)
- Research to determine and implement the appropriate sensor for the Detector (ultrasonic sensor, infrared sensor, **magnetic sensor**, load sensor…)
- Research to determine and implement the appropriate RF value and module to provide communication between nodes for the project (315Mhz, 433Mhz, **2.4Ghz**…)
- Research and implement LED RGB, seven-segment LED
- Research and implement real-time communication channel
- Research and host Web API on a cloud service
- Study and develop a mobile application (**Android**, iOS, Windows phone…)
- Study and develop program using embedded language (**Arduino**, C, **C++**, **Python**, Java Embedded, C#…)
- Study and create a Web API (Spring MVC, **ASP.NET**, Ruby…)
- Study and create a database (**SQL**, Oracle, MySQL, **SQLite** …)

B. Software – Hardware Project Management Plan

1. Problem Definition

1.1. Name of this Capstone Project

- Official name: Parking Guidance System Solution
- Vietnamese name: Giải pháp hệ thống chỉ dẫn đỗ xe
- Abbreviation: PGSS

1.2. Problem Abstract

As the economy of Vietnam growth, the number of personal cars also increasing, and that create a high proportion of traffic generated by drivers seeking vacant sparking lots. The current common Parking Guidance Systems in Vietnam are only suitable for a small number of indoor parking areas, and can't be implemented for outdoor parking areas, because of the need of complicated planning and wiring. Moreover, all of the current PGS parking areas are working separately in their own local area network so there is no way for drivers to know the current available parking lots except by coming to the entrance.

We provide a system which ease the complicated in set up a PGS for parking areas. Furthermore, we make the system in a way that each PGS parking area can connect to each other so we can provide more information to drivers to help them find a suitable parking areas more quickly and easily.

1.3. Project Overview

1.3.1. Current Situation

In the market, we have some ways to manage car park:

- The guard check each car in each parking lot:
  - Advantages:
    - Can check empty or using slot exactly
  - Disadvantages:
    - Consume people's energy
    - Need much time to check all the parking lot
- Tradition PGS base on RS-485:
  - Advantages:
    - Can automatically check the empty slot in car park
  - Disadvantages:
    - Limited number of managed parking lots
    - Complicated wired system
    - Limited information provided to drivers

1.3.2. The Proposed System

Based on the result of our research, we propose the following solution: A Parking Guidance

System based on Internet of Things that utilize the RF wireless technology to communicate between components. The system consists of Detectors, Indicator LED, Information LED Displays, Reservation Barrier, Central Control Unit, Web API Server and a Mobile Application.

After the Detector detect a car in the parking lot, it will send a signal to the Central Control Unit (CCU). The CCU will command the Indicator LED above that parking lot change to occupied color, update all related Information LED Displays, send a message to Web API Server to update information of parking lot on the server and Mobile Application.

In case of users want to reserve a parking lot in the parking area, they can use the Mobile Application to make a reservation. The Web API Server will update the database based on the reservation and the Reservation Barrier will lock the reserved parking lot.

1.3.2.1. Interaction Block

- This block will be place in each parking lot to check existed car and control signal light, barrier.
- Arduino is the main board to control the Interaction block, which show information to the end customer.
- Ultrasonic sensor is used to check the existed car in each parking lot
- DC Servo to monitor the barrier.
- 12A DC-to-DC step down module used to convert voltage high-to-low for other hardware.

1.3.2.2. Information Block

- This is the led panel in each area to show the number of empty slot in each area.
- Arduino is also the main board to control this block.

1.3.2.3. Central Control Unit

- This will be the central point of all Interaction Module.
- It will be control be Raspberry Pi 3.
- Send and receive data from server to analyze then monitor the Interaction Block.

1.3.2.4. Web API Server

- ASP.NET API to communicate between the mobile app, database and CCU.
- Get the position of each car park base on address to view on mobile app.

1.3.2.5. Mobile Application

- Our priority OS for Mobile Application is Android because of a higher market share than other mobile OS and a lower barrier of entry
- The App utilizes the Google Map API to provide an interactive map that show in real time the available parking lots in each parking areas.

1.3.3. Boundaries of the System

- System is available for both manager and end user.
- The language of system is English
- The input and output of system is car slot

- The boundaries of mechanical parts include:
  - End user need to park their car correctly in parking lot.
  - The information, which show number of empty slot may delay 5-10 seconds when the system start.
  - The system will run correctly when the weather condition is good.
- The boundaries of mobile application include:
  - End user only pay for the booking time, not pay the parking time on mobile app.
  - Need connect to internet to run smoothly.
- The complete product includes:
  - The entire PGSS system in car park.
  - The mobile application for manager and end user.
  - The database to store all needed information.
  - All the documents of the project.

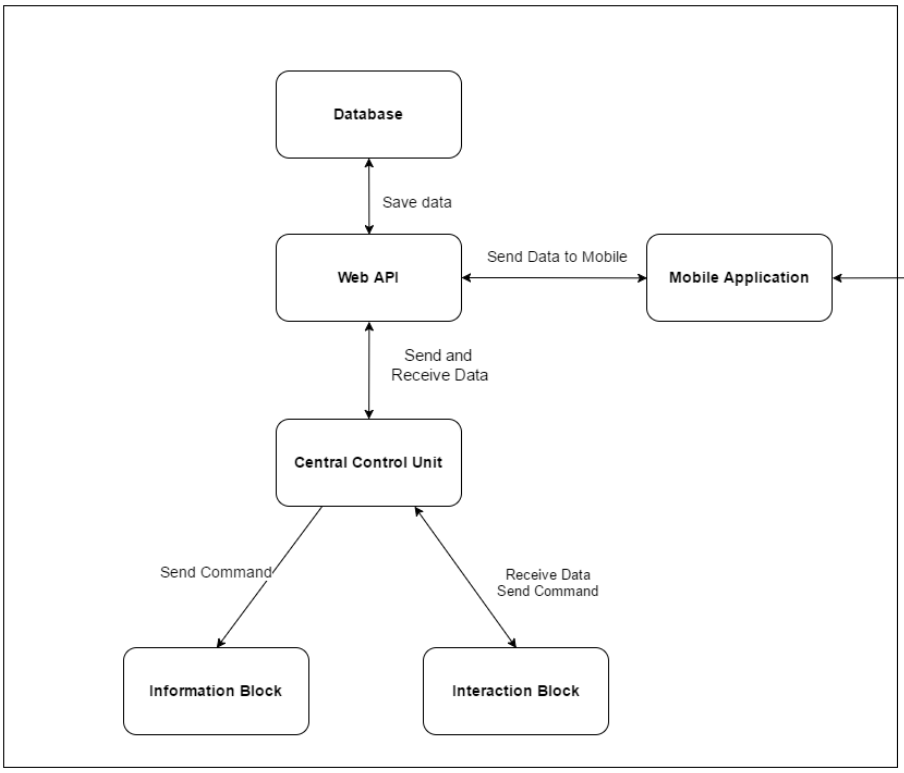


Figure 5: Project Block Diagram

1.3.4. Future Plans

There are no perfect solutions to problems, as well as there are no perfect systems. With the inexperience of our team members and the time constrains, our proposed solution and project contains many issues. Below are the problems encountered in this project:

- **Parking Operate Knowledge:** We are not experts in parking operating. All functions and features are developed in order to serve the needs which we had identified during 4 months of research.

- **Hardware Knowledge:** We are inexperienced with hardware. All the hardware components chosen to be used in this project is based on our familiar with them, or based on the shortest time we need to learn how to use them. So they are only the most appropriate, not the best choice for the project.
- **Single point of failure:** The communication of the PGS system and server is highly depended on the Central Control Unit in each parking area. So if the Central Control Unit crash, the PGS can no longer commute with server.
- **Server crash:** All the needed data for the app is stored in the server. So if server crash, all the devices cannot get parking area information.
- **Security:** Currently, there is few possible problems encountered with RF, as RF is vulnerable to replay attack.

Our future plan is try to solve these problems one by one. We design the system with separated modules in mind to make it easy to change one module without affect others and we also make it easy to scale to bigger models.

1.3.5. Development Environment

1.3.5.1. Hardware requirements

For Web API Server

Components	Requirement
DTU	10 DTU
Storage	250 GB

Table 3: Database requirement

Components	Requirement
Number of cores	1 core
RAM	1.75 GB Ram
Storage	10 GB

Table 4: API Service Requirement

For CCU

Components	Hardware
Mainboard	Raspberry Pi 3
Communication	USB Cable
Sensor Devices	Magnetometer
Motors	Servo
Power Source	

Table 5: Provide CCU Hardware

1.3.5.2. Software requirements

- Windows XP/7/8/10: operating system for developing and deploying.
- SQL Server Express 2012: used to create and manage database for PGSS.
- Visual Studio 2015: used to develop API.
- Arduino IDE: used to develop Arduino program.
- Proteus 8: used to drawing board with other hardware.
- Github & SourceTree: used for source control.
- StarUML: used to create models and diagrams.
- Slack: used for communication and meeting.
- C/C++: used for embedded module
- Python 3: used for Central Control Unit
- C#: used for web server
- Java: used for mobile application

2. Project organization

2.1. Software Process Model

This project is developed under Iterative and incremental development model. We apply customized Iterative and incremental development model to capable with current situation in our team. We choose this model because of the following reasons:

- We are still inexperienced and by develop the system through iterations (repeated cycles) and incrementally (in small portions of time), we can learn from our mistakes and apply that knowledge on the next iteration.
- We are researching and developing the system at the same time, so using this model allow us more flexibility to adapt to changes.
- Working with embedded system hides a lot of problems that are unknown in the planning phase until it is too late. With Iterative and incremental development model, we test the system in small portion at a time, therefore reduce risk and build a feature rich and robust system.

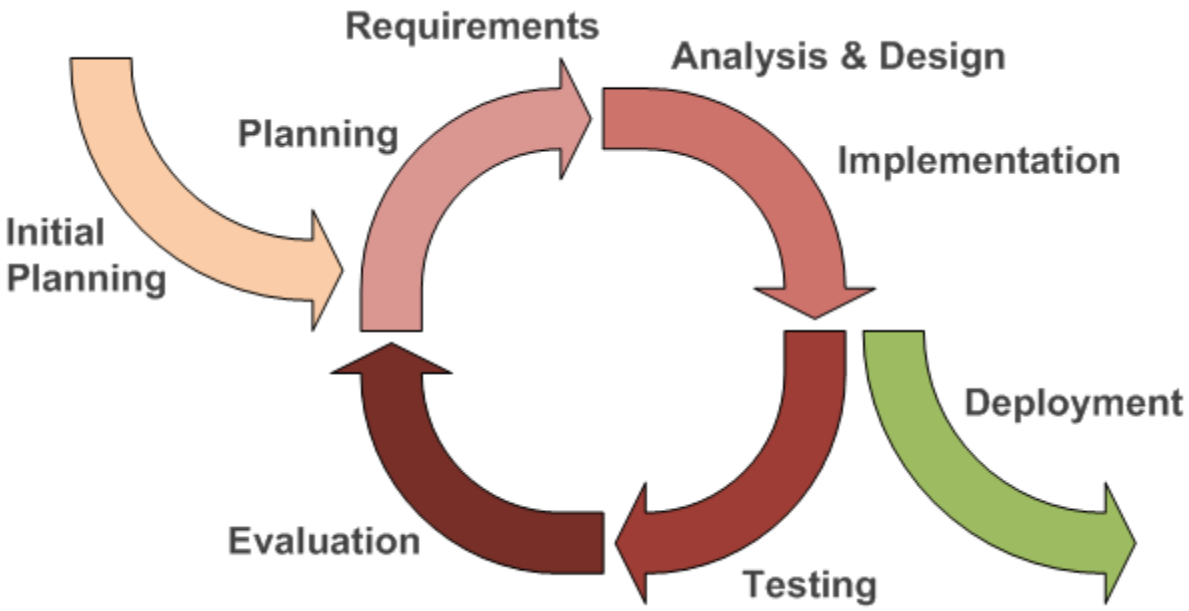


Figure 6: Iterative and Incremental development

2.2. Roles and responsibilities

No	Full name	Team Role	Responsibilities
1	Nguyễn Đức Lợi	Supervisor, Project Manager	• Specify user requirement • Advisor for ideas and solutions • Control the development process • Give out techniques and business analysis support
2	Trần Nguyễn Minh Trung	Team Leader, BA, Developer, Tester	• Managing process • Managing budget • Dividing tasks for team member • Create test plan • Clarifying requirements • Prepare document • Coding • Testing
3	Bùi Phú Hiệp	Team Member, Developer, Tester	• Create test plan • Clarifying requirements • Prepare document • Coding • Testing

4	Nguyễn Đỗ Phương Huy	Team Member, Developer, Tester	• Create test plan • Clarifying requirements • Prepare document • Coding • Testing

Table 6: Roles and Responsibilities Details

2.3. Tools and Techniques

Tools	
Operating System	Windows 7 Ultimate
	Raspbian Jessie
Developing tool	Android Studio
	Visual Studio 2015 Community
	IDLE 3
	Arduino IDE 1.6.12
Managing Database	SQL Server 2014 Management Studio
Source Control	Git 2.8.1 (Server https://github.com)
	SourceTree 1.9.10
Communication tool	Gmail
	Slack
	Trello
Models and Diagrams tool	https://www.draw.io/, StarUML

Table 7: Tools

Techniques	
Embedded System	C/C++
	Arduino

	Python 3
Mobile System	Android SDK
	Retrofit 2
	Google Map
Web Server System	Azure Cloud
	ASP.NET
Database Management System	SQL Server 2014
	SQLite 3.7

Table 8: Techniques

3. Project Management Plan

3.1. System development life cycle

Incremental development slices the system functionality into increments (portions). In each increment, a slice of functionality is delivered through cross-discipline work, from the requirements to the deployment. The Unified Process groups increments/iterations into phases:

Phase	Description	Deliverables	Risks
Inception	In this phase, we will identify project scope, requirements (functional and non-functional) and risks at a high level but in enough detail that work can be estimated	• Introduction of proposed system • Software and Hardware requirement specification • Project Task Plan and Risk	• The lack of knowledge may lead to misunderstand of the requirement • The inexperienced of team may lead to deficient in Task Plan and Risk
Elaboration	Delivers a working architecture that mitigates the top risks and fulfills the non-functional requirements	• Software and Hardware design document	• System architecture or design issues may arise because not all requirements are gathered
Construction	Incrementally fills-in the architecture with production-ready code produced from analysis, design, implementation, and testing of the functional requirements	• Completed and fully tested system • Implementation and Test document	• The lack of knowledge of team member about hardware • The inexperienced of team may lead to missing test cases • There may be hidden High to Critical bugs in the system
Transition	Delivers the system into the production operating environment	• User manual document • Installation guide document • The final and full version document of the system	• The documents may not fully describe the system

Table 9: System Development Life Cycle

Each of the phases may be divided into 1 or more iterations, which are usually time-boxed rather than feature-boxed.

3.2. Plan Detail

Iteration	Scope	Evaluation	Activities	Estimated Duration	Assign Responsibilities
Initial Iteration	Initial team workplace and identify project scope	A working team environment	• Set up Git Repository with Gitflow • Set up Slack • Set up Trello	5 days	TrungTNM, HiepBP, HuyNĐP
Iteration 1	Identify boundaries of the system, planning software and hardware. Create a proof-of-concept prototype.	Report 1, Report 2 and a proof-of-concept prototype	• Introduction document • Software and Hardware Project Management Plan document • Proof-of-concept prototype	15 days	TrungTNM, HiepBP, HuyNĐP
Iteration 2	Produce an architectural prototype	Report 3, Report 4 and an architectural prototype	• Software and Hardware Requirement Specification document • Software and Hardware Design Description document • Architectural prototype	15 days	TrungTNM, HiepBP, HuyNĐP

Iteration 3	Build the product (up to beta release)	Report 5 and a working product (beta release)	• System Implementation and Test document • PCB • Mobile Application • Web API Server	15 days	TrungTNM, HiepBP, HuyNĐP
Iteration 4	Finish the product (full product release)	Report 6 and the completed product	• Software and Hardware User's Manual document • Product demonstration model	15 days	TrungTNM, HiepBP, HuyNĐP
Final Iteration	Prepare for Demo Day	Final Documentation, Presentation Slide	• Final Document • Mini Document • CD contains all source code • Presentation Slide	5 days	TrungTNM, HiepBP, HuyNĐP

Table 10: System Development Detail Plan

3.3. All Meeting Minutes

All meeting minutes are saved at:

https://github.com/Hinaka/-FPT-CAPSTONE-PGSS/tree/master/Common

4. Coding Convention

4.1. C/C++ Convention

C/C++: Using to develop program and solve algorithm on hardware.

Summary:

- Naming Convention:
  - Using Pascal case for class name.
  - Using Camel case for function, variable's name.
  - The #define and global variable's name must uppercase and separate by underscore. Ex: GLOBAL_VARIABLE.
- Commenting Convention:
  - Place the comment on the separate line with function.
  - Place the comment at the end of the line, which has calculation formula.

More details about coding conventions for C/C++ language by Google:

https://google-styleguide.googlecode.com/svn/trunk/cppguide.html

4.2. C#, ASP.NET Convention

C#: Using to develop Web API

Summary:

- Naming Convention:
  - Use Camel case for variable's name.
  - Use Pascal case for class's name, function's name.
  - Global variable's name must uppercase and separate by underscore.

More detail about code conventions for C# language by Microsoft:

https://msdn.microsoft.com/en-us/library/ff926074.aspx

4.3. Python Convention

Python: Using to develop program on Raspberry Pi

Summary:

- Naming Convention:
  - "Internal" means internal to a module or protected or private within a class.

- Prepending a single underscore (_) has some support for protecting module variables and functions (not included with import * from). Prepending a double underscore (__) to an instance variable or method effectively serves to make the variable or method private to its class (using name mangling).
- Place related classes and top-level functions together in a module. Unlike Java, there is no need to limit yourself to one class per module.
- Use Pascal Case for class names, but lower_with_under.py for module names.

More detail about code conventions for Python by Google:

https://google.github.io/styleguide/pyguide.html

4.4. Android Convention

Android use Java and HTML to develop mobile application

Summary:

- Naming convention:
  - Follow basic principle of <WHAT>_<WHERE>_<DESCRIPTION>_<SIZE> for resource names
  - Follow basic principle of <WHAT>_<WHERE>.XML for layout
  - Follow basic principle of <WHERE>_<DESCRIPTION> for string resources
  - Follow basic principle of <WHERE>_<DESCRIPTION>_<SIZE> for drawable resource
  - Follow basic principle of <WHAT>_<WHERE>_<DESCRIPTION> for IDs

More detail about code conventions for Android by Google and our team:

https://source.android.com/source/code-style.html

https://github.com/Hinaka/-FPT-CAPSTONE-PGSS/tree/master/Common

C. Software – Hardware Requirement Specification

1. Software Requirement Specification

1.1. Software Requirement

Manager can show the information of their car park to the end user, which will increase the interaction between car park provider and end user. The information includes:

- Address
- Contact info
- Number of empty parking lot

End user can find the nearest car park, which has empty parking lot.

Manager can manage their car park easily; make an automatic system to guide the end user base on the interaction panel, which show number of empty parking lot in each area and the status light on each parking lot.

Users can see empty slot and detail information about parking area by touching a marker on map.

User can reserve a parking slot.

1.2. GUI Requirement

User interface of mobile app must be simple, clearly and easy to use.

The color of mobile app must be elegant, not garish.

Each UI element must be arranged logically, allowing user access easily.

Meet all main function requirements.

2. Hardware Requirement Specification

2.1. Hardware Requirement

2.1.1. Hardware Interface

The hardware interface must have satisfied the following requirements:

- Easy to replace
- Low-cost module
- Easy to implement

Based on project requirement we have choose following hardware components.

2.1.1.1. Block Diagram

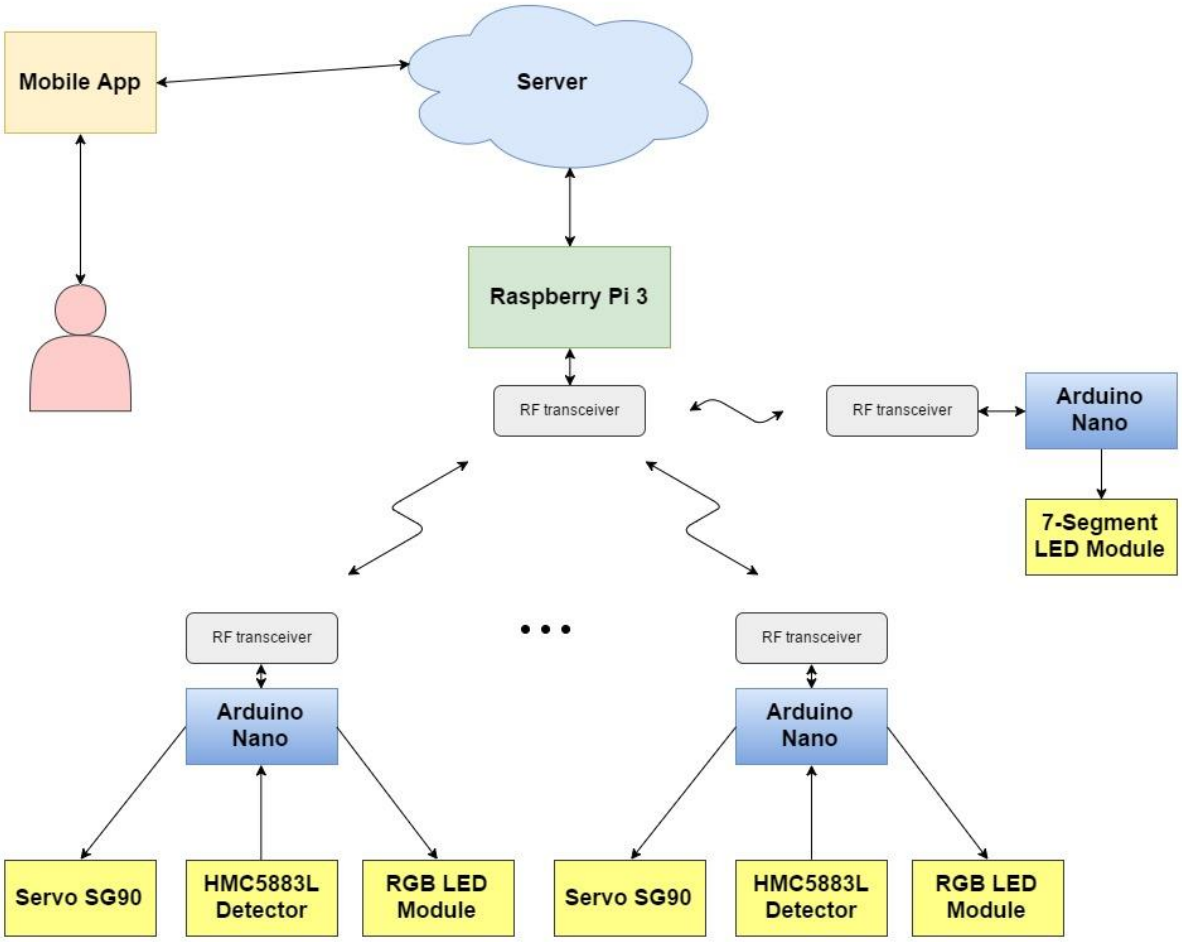


Figure 7: PGSS Block Diagram

This block diagram shows the basic concept IoT provided in the project. The Raspberry Pi 3 will play as Hub role, and Arduino Nano will play as Node role, which communicate with Hub through RF transceiver. The Hub also connect to a Cloud Server, provides the ability for User to interactive with through a Mobile App that also connect to the same Cloud Server.

2.1.1.2. Raspberry Pi 3

Figure 8: Raspberry Pi 3

Overview: To communicate with all other hardware component and processing value, we must have a Central control unit, there are many kind of central control unit in the market. After evaluate requirement of project, we decide to choose Raspberry Pi 3. Raspberry Pi 3 is powerful mini-computer with many features.

Specification:

SoC	Broadcom BCM2837
CPU	4× ARM Cortex-A53, 1.2GHz
GPU	Broadcom VideoCore IV
RAM	1GB LPDDR2 (900 MHz)
Network	10/100 Ethernet, 2.4GHz 802.11n wireless
Bluetooth	Bluetooth 4.1 Classic, Bluetooth Low Energy
Storage	microSD
GPIO	40

Table 11: Raspberry Pi 3 – Specification

More details about Raspberry Pi 3:

https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

2.1.1.3. Arduino Nano

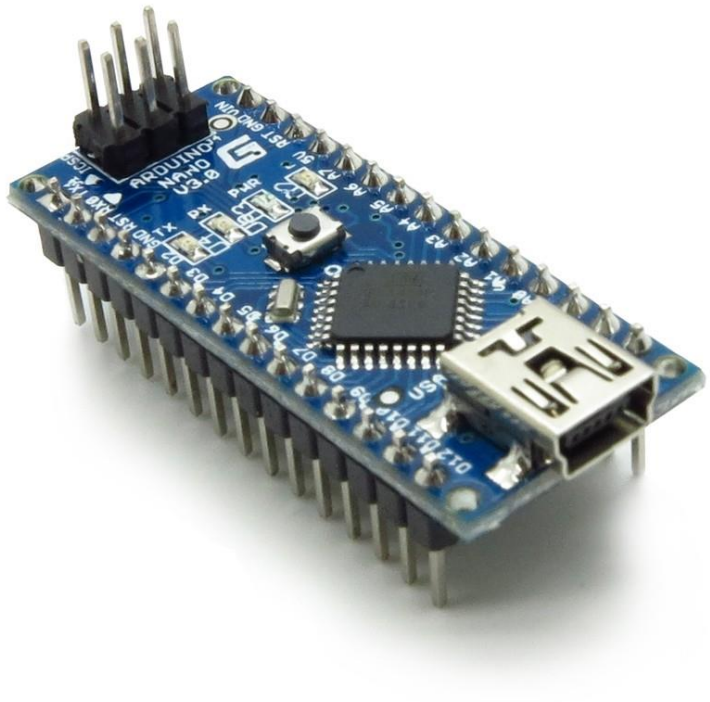


Figure 9: Arduino Nano

Overview: The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 (Arduino Nano 3.x).

Specification:

Microcontroller	ATmega328
Architecture	AVR
Operating Voltage	5 V
Flash Memory	32 KB of which 2 KB used by bootloader
SRAM	2 KB
Clock Speed	16 MHz

Analog I/O Pins	8
EEPROM	1 KB
DC Current per I/O Pins	40 mA (I/O Pins)
Input Voltage	7-12 V
Digital I/O Pins	22
PWM Output	6
Power Consumption	19 mA
PCB Size	18 x 45 mm
Weight	7 g
Product Code	A000005

Table 12: Arduino Nano - Specification

More detail about Arduino Nano:

https://www.arduino.cc/en/Main/arduinoBoardNano

2.1.1.4. Compass Module 3-Axis HMC5883L

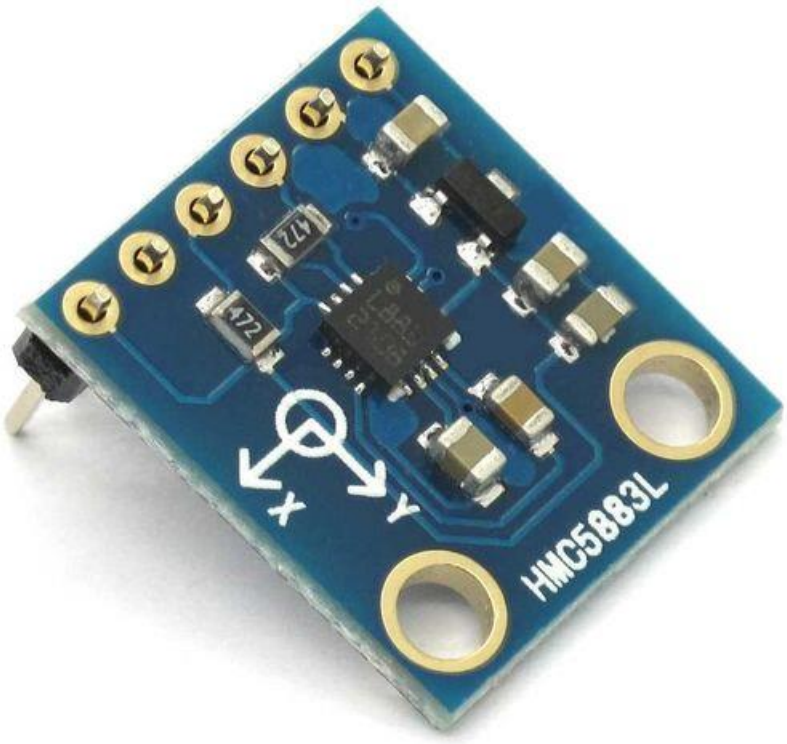


Figure 10: Compass Module 3-Axis HMC5883L

Overview: For detecting obstacle, we choose The Compass Module 3-Axis HMC5883L instead of ultrasonic sensor because ultrasonic sensor has many weaknesses, they are not accuracy, cannot be used outdoor in the bad weather in Vietnam.

The Compass Module 3-Axis HMC5883L is a low-field magnetic sensing device with a digital interface.

We choose The Compass Module 3-Axis HMC5883L because:

- It has reasonable price.
- Compatible with arduino and other board.
- Compact size.

Specification:

Input and Output Pins:

Pin		I/O	Function
Name	**No.**		
VIN	1		Supply Voltage – 2.7 to 6.5 VDC
GND	2		Ground
SCL	3	I	I^2C Clock
SDA	4	IO	I^2C Data
RDY	5	I	Data Ready

Table 13: The Compass Module 3-Axis HMC5883L - Pin Function

2.1.1.5. RF module nRF24L01+

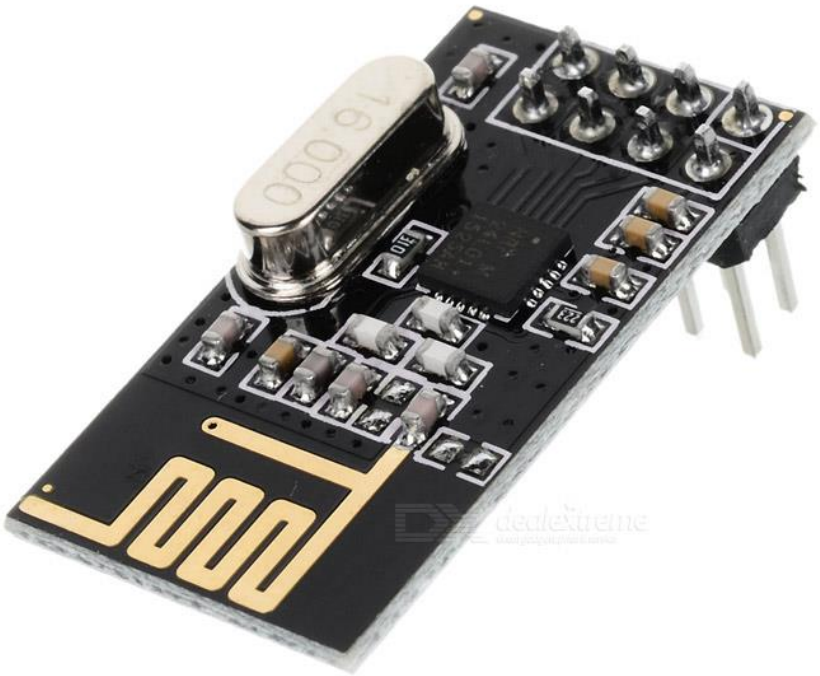


Figure 11: RF module nRF24L01+

Overview: Reason for PGSS use RF module nRF24L01 to communicate between Central control unit and other hardware component:

- It has reasonable price.
- Easy to buy.
- Ultra low power consumption.

Specification:

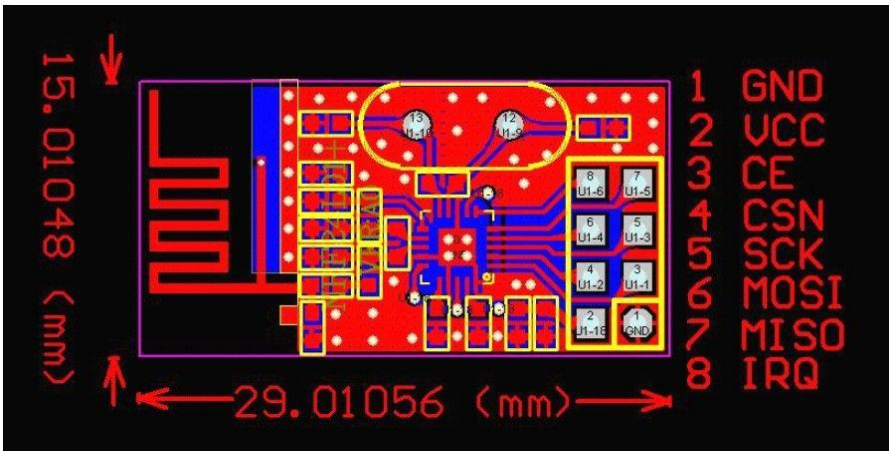


Figure 12: RF module nRF24L01+ - Specification

Pin		I/O	Description
No.	Name		
1	GND		Power Supply Ground
2	VCC		3.3V
3	CE	I	Chip Enable
4	CSN	I	SPI Chip Select
5	SCK	I	SPI Clock
6	MOSI	I	SPI Slave Data Input
7	MISO	O	SPI Slave Data Output
8	IRQ	O	Maskable Interrupt Pin

Table 14: RF Module nRF24L01 – Pin function

2.1.1.6. Information LED Display Module

Information LED Display Module include: 7-segment LED Display, TPIC6B595 Power Logic 8-Bit Shift Register

7-segment LED Display

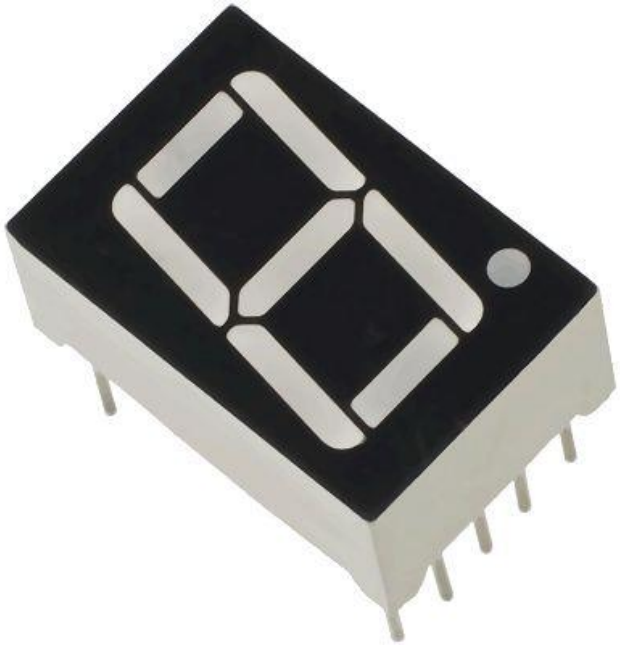


Figure 13: 7-segment LED Display

Specification:

- 0.56 inch digit height
- Super Red emitting color
- White segment color, gray face
- Low current operation
- Easy mounting on PCB boards or sockets

TPIC6B595 Power Logic 8-Bit Shift Register

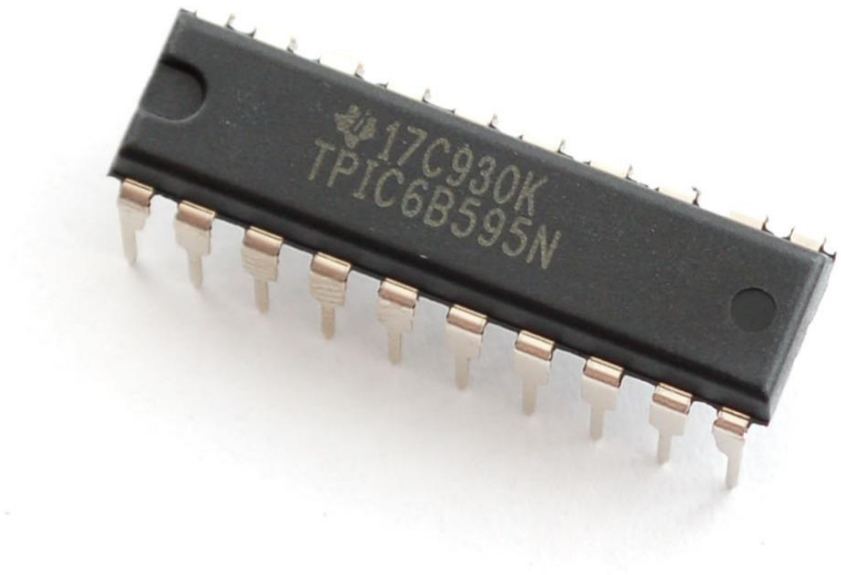


Figure 14: TPIC6B595 Power Logic 8-Bit Shift Register

Specification:

To display high power 7-segment display, we choose IC TPIC6B595 instead of IC 74HC595 because TPIC6B595 is a simple shift register IC that can control high-voltage/high-current devices directly**.** Each output pin can sink 150mA and then supports the maximum of Load Voltage at 50V.

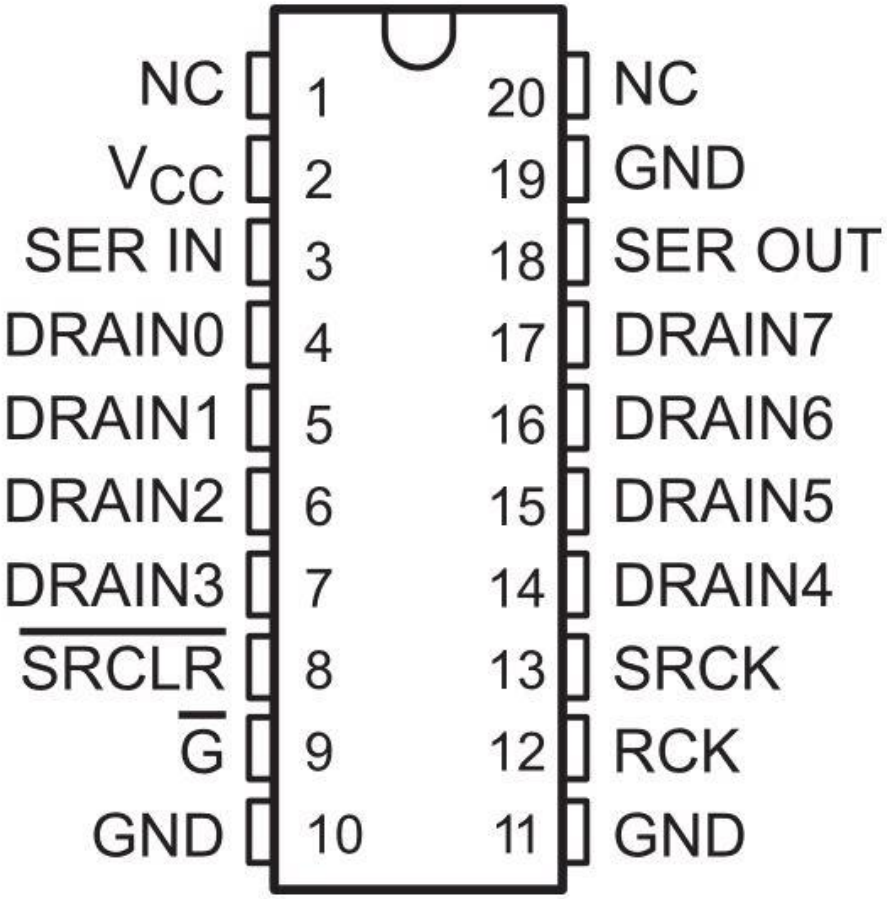


Figure 15: TPIC6B595 Pin-outs

Pin		I/O	Description
Name	**No.**		
DRAIN0	4	O	Open-drain output
DRAIN1	5		
DRAIN2	6		
DRAIN3	7		
DRAIN4	14		
DRAIN5	15		
DRAIN6	16		
DRAIN7	17		
G	9	I	Output enable, active-low

GND	10,11,19	-	Power ground
NC	1, 20	-	No internal connection
RCK	12	I	Register clock
SERIN	3	I	Serial data input
SEROUT	18	O	Serial data output
SRCK	15	I	Shift register clock
SRCLR	8	I	Shift register clear, active-low
VCC	2	I	Power supply

Table 15: IC TPIC6B595 - Pin Function

2.1.1.7. Indicator LED Module

Indicator LED Module include: Common anode RGB LED, TIP122 Transistor

RGB LED common anode

Figure 16: RGB LED common anode

Overview:

RGB LED allows you to change the lights to any color to show state of parking slot.

Specification:

- Forward Voltage (RGB): (2.0, 3.2, 3.2)V
- Max Forward Current (RGB): (20, 20, 20)mA
- Max Luminosity (RGB): (2800, 6500, 1200)mcd

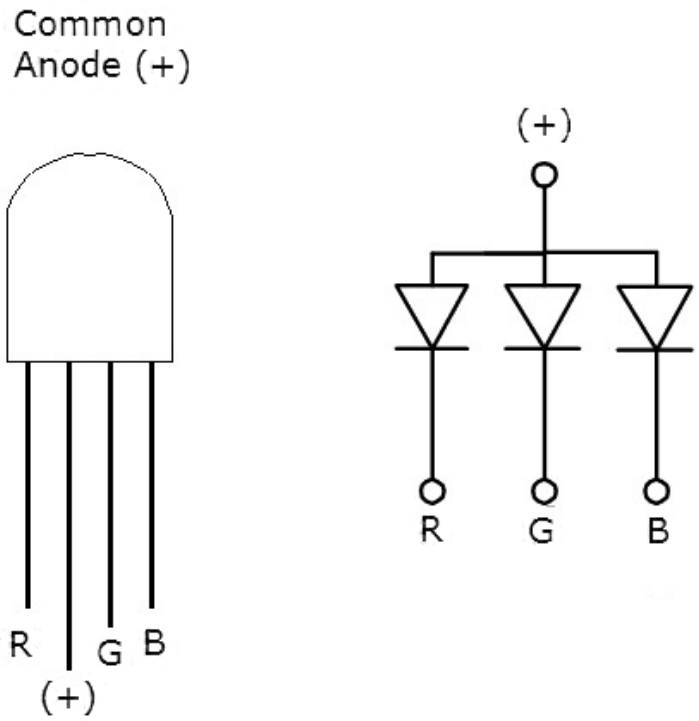


Figure 17: RGB LED common anode pin-out

TIP122 Transistor

Figure 12:: TIP122 Transistor

Overview: A single digital pin on Arduino Nano do not provide enough current to power RGB LED, A solution for this situation is to use an NPN Darlington Transistor designed for medium power linear switching applications, so we use TIP122 Transistor to provide RGB LED with power from an external source. It can power devices up to 100VDC at 5 Amps.

Specification:

- TIP122 is power transistors
- Collector Current: 5 ampere
- Collector-Emitter Volt: 100 volts
- Power Dissipation: 65 watts

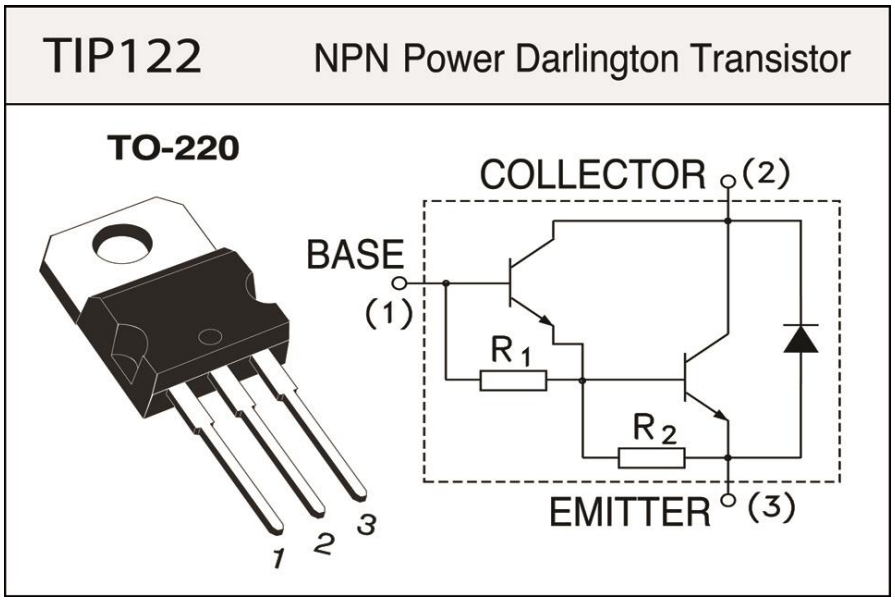


Figure 12: TIP122 Transistor- Pin Layout

2.1.1.8. Servo Motor SG90

Figure 13:: Servo Motor – Tower Pro SG90

Overview: PGSS use Servo Motor SG90 to control barrier at each parking slot.

Specification:

Torque	1.80 kg-cm at 4.8V
Speed	0.1sec/60° at 4.8V
Voltage	4.0V to 7.2V, 4.6V - 5.2V nominal
Dimensions	23mm x 12.2mm x 29mm
Rotation range	180°
Weight	9g
Pulse width	500-2400uS
Operating Temperature range	30°C to 60°C

Table 7: Servo Motor SG90 – Specification

Pin of Servo SG90	Name	Description
Red	VCC	Power supply 5V
Black	GND	Power supply ground
Yellow	Signal	The servo will move based on the signal sent to signal wire.

Table 16: Servo Motor SG90 – Pin-outs

2.1.2. Communication Protocol

- We communicate between hardware component and board through GPIO pin.
- Arduino Nano board communicate with Raspberry Pi 3 via RF Module.

2.2. System Overview Use Case

Figure 18: Overview Use Case Diagram

2.3. List of Use Case

2.3.1. Manager Use Case

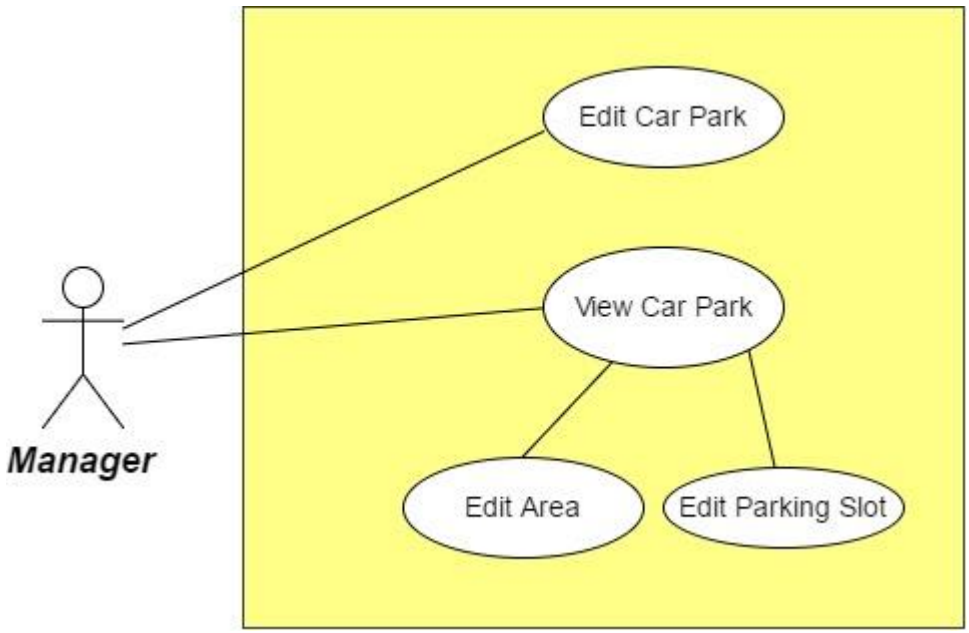


Figure 19: Manager Use Case Diagram

Use case specifications

Use Case-1 specification			
Use-case no.	PGSS01	**Use-case version**	1.0
Use-case name	Edit Car Park		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:
- Manager

Summary:
- This use case allow user to change the configuration of their system.

Goal:
- Manager can change the information of car park, which show to the end user.

Triggers:
- User click on once car park button.

Preconditions:
- Mobile application is already launch.
- Manager has been logged in

Post Conditions:
- **On Success:** New configuration is apply and save to server
- **On Failure:** Show error message

Main Success Scenario:

No.	**Actor Action**	**System Response**

1	User select once car park	Show the car park detail information on mobile
2	User click on “Setting” button	Application navigate to “Setting” menu
3	User select option in the Menu Change by click toggle or change value in the text box Select “Submit” button	Change the value and save to server

Alternative Scenario:
- N/A

Exceptions:
- N/A

Business Rules:
- N/A

Use Case-2 specification			
Use-case no.	PGSS02	**Use-case version**	1.0
Use-case name	View Car Park		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:
- Manager

Summary:
- This use case allow user to view their car park info.

Goal:
- User can view the information of car park.

Triggers:
- User select their car park.

Preconditions:
- Mobile application is already launch.
- Manager has been logged in

Post Conditions:
- **On Success:** User can view the information of car park.
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select once car park	Show the car park detail information on mobile

Alternative Scenario:

- N/A

Exceptions:

- N/A

Business Rules:

- N/A

Use Case-3 specification			
Use-case no.	PGSS03	**Use-case version**	1.0
Use-case name	Edit Area		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:

- Manager

Summary:

- This use case allow user to change the status of each area.

Goal:

- The status of selected area updated and change in mobile app.

Triggers:

- User select their car park.
- User select area in selected car park.

Preconditions:

- Mobile application is already launch.
- Manager has been logged in

Post Conditions:

- **On Success:** New configuration is apply and save to server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select car park	Application change to car park detail page
2	User select area in the selected car park	Application change to area detail page
3	User select status in the drop down list. Click “Update” button	The status of the area will change on server and update in mobile application

Alternative Scenario:

- N/A

Exceptions:

- N/A

Business Rules:

- N/A

Use Case-4 specification			
Use-case no.	PGSS04	**Use-case version**	1.0
Use-case name	Edit Parking Slot		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:

- Manager

Summary:

- This use case allow user to manage the parking slot.

Goal:

- The status of selected area updated and change in mobile app.

Triggers:

- User select their car park.
- User select area in selected car park.
- Then select parking slot

Preconditions:

- Mobile application is already launch.
- Manager has been logged in

Post Conditions:

- **On Success:** New configuration is apply and save to server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select car park	Application change to car park detail page
2	User select area in the selected car park	Application change to area detail page
3	User select parking slot to edit After change information, select "Update" button	The information of parking slot is change on server and update in mobile application.

Alternative Scenario:

No.	Actor Action	System Response
1	User select car park	Application change to car park detail

		page
2	User select area in the selected car park	Application change to area detail page
3	User click menu beside list parking spot to delete.	The parking spot will be set to deleted in server and update in mobile app.

Exceptions:
- N/A

Business Rules:
- N/A

2.3.2. Administrator Use Case

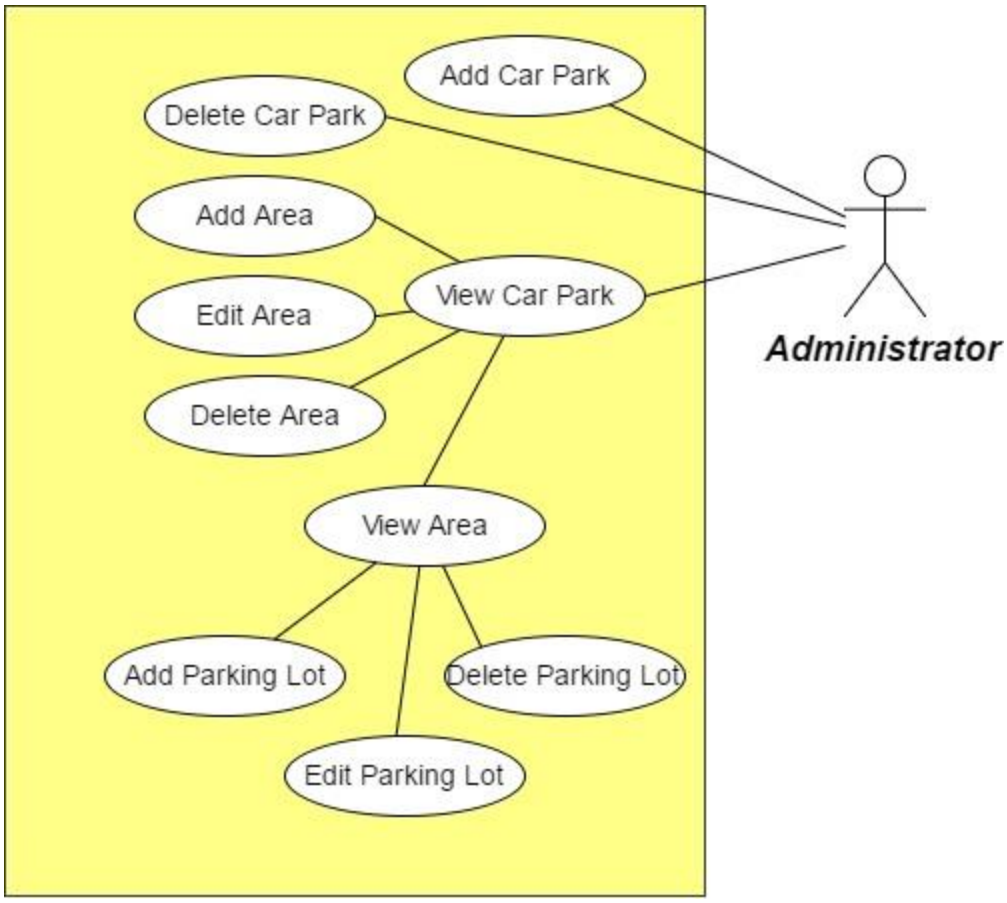


Figure 20: Manager Use Case Diagram

Use Case-5 specification			
Use-case no.	PGSS05	**Use-case version**	1.0
Use-case name	Add Car Park		

Author	Bui Phu Hiep		
Date	13/02/17	Priority	High

Actor:
- Administrator

Summary:
- This use case allow user to add new car park to the system

Goal:
- New car park is added and save to server.

Triggers:
- User click on "Add" button.

Preconditions:
- Mobile application is already launch.
- Administrator has been logged in

Post Conditions:
- **On Success:** New car park is save to server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User click on "Add" or "+" button	Application navigate to add car park menu
2	User fill in the textbox Select "Submit" button	New car park with filled in info is added to server

Alternative Scenario:
- N/A

Exceptions:
- Name of the car park is unique
- Address of the car park is unique (don't has same latitude and longitude)

Business Rules:
- N/A

Use Case-6 specification			
Use-case no.	PGSS06	Use-case version	1.0
Use-case name	Delete Car Park		
Author	Bui Phu Hiep		
Date	13/02/17	Priority	High

Actor:
- Administrator

Summary:
- This use case allow user to delete a car park on the system

Goal:

- Delete a car park on the server

Triggers:

- User click on "x" button.

Preconditions:

- Mobile application is already launch.
- Administrator has been logged in

Post Conditions:

- **On Success:** Delete a car park on server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User click on "Remove" or "x" button	Application show the confirm button
2	User click "Accept"	Car park is removed from the server

Alternative Scenario:

- N/A

Exceptions:

- **N/A**

Business Rules:

- N/A

Use Case-7 specification			
Use-case no.	PGSS07	**Use-case version**	1.0
Use-case name	View Car Park		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:

- Manager

Summary:

- This use case allow user to view car park info.

Goal:

- User can view the information of car park.

Triggers:

- User select the car park.

Preconditions:

- Mobile application is already launch.
- Manager has been logged in

Post Conditions:

- **On Success:** User can view the information of car park.
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response

1	User select once car park	Show the car park detail information on mobile

Alternative Scenario:
- N/A

Exceptions:
- N/A

Business Rules:
- N/A

Use Case-8 specification			
Use-case no.	PGSS08	**Use-case version**	1.0
Use-case name	Add Area		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:
- Administrator

Summary:
- This use case allow user to add new area to the system

Goal:
- New area is added and save to server.

Triggers:
- User select car park
- User click on “Add” button.

Preconditions:
- Mobile application is already launch.
- Administrator has been logged in

Post Conditions:
- **On Success:** New area is save to server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select a car park	Show the car park detail information screen
2	User click on “Add” or “+” button	Application navigate to add area menu
3	User fill in the textbox Select “Submit” button	New area with filled in info is added to server

Alternative Scenario:

- N/A

Exceptions:

- N/A

Business Rules:

- N/A

Use Case-9 specification			
Use-case no.	PGSS09	**Use-case version**	1.0
Use-case name	Edit Area		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:

- Administrator

Summary:

- This use case allow user to update to the system

Goal:

- New value is updated to server.

Triggers:

- User select car park
- User click on “Edit” button next to an area.

Preconditions:

- Mobile application is already launch.
- Administrator has been logged in

Post Conditions:

- **On Success:** New value is saved to server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select a car park	Show the car park detail information screen
2	User click on “Edit” button	Application navigate to edit area menu
3	User fill in the textbox Select “Submit” button	New value of area with filled in info is updated to server

Alternative Scenario:

- N/A

Exceptions:

- N/A

Business Rules:

- N/A

Use Case-10 specification			
Use-case no.	PGSS10	**Use-case version**	1.0
Use-case name	Delete Area		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:
- Administrator

Summary:
- This use case allow user to delete an area on the system

Goal:
- Delete an area on the server

Triggers:
- User select a car park
- User click on "x" button next to the area.

Preconditions:
- Mobile application is already launch.
- Administrator has been logged in

Post Conditions:
- **On Success:** Delete an area on server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select once car park	Show the car park detail information on mobile
2	User click on "Remove" or "x" button next to the area	Application show the confirm button
3	User click "Accept"	Area is removed from the server

Alternative Scenario:
- N/A

Exceptions:
- **N/A**

Business Rules:
- N/A

Use Case-11 specification			
Use-case no.	PGSS11	**Use-case version**	1.0
Use-case name	View Area		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:
- Manager

Summary:
- This use case allow user to view area info.

Goal:
- User can view the information of area.

Triggers:
- User select the car park.
- User select the area in car park

Preconditions:
- Mobile application is already launch.
- Manager has been logged in

Post Conditions:
- **On Success:** User can view the information of area.
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select once car park	Show the car park detail information on mobile
2	User select area in the list of area	Show the area information on mobile

Alternative Scenario:
- N/A

Exceptions:
- N/A

Business Rules:
- N/A

Use Case-12 specification			
Use-case no.	PGSS12	**Use-case version**	1.0
Use-case name	Add Parking Lot		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:
- Administrator

Summary:
- This use case allow user to add new parking lot to the system

Goal:
- New parking lot is added and save to server.

Triggers:
- User select area
- User click on “Add” button.

Preconditions:
- Mobile application is already launch.
- Administrator has been logged in

Post Conditions:
- **On Success:** New parking lot is saved to server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select a car park	Show the car park detail information screen
2	User select a area	Show the area information screen
3	User click on "Add" or "+" button	Application navigate to add parking lot screen
3	User fill in the textbox Select "Submit" button	New parking lot with filled in info is added to server

Alternative Scenario:
- N/A

Exceptions:
- N/A

Business Rules:
- N/A

Use Case-13 specification			
Use-case no.	PGSS13	**Use-case version**	1.0
Use-case name	Edit Parking Lot		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:
- Administrator

Summary:
- This use case allow user to update parking lot to the system

Goal:
- New value of selected parking lot is updated and save to server.

Triggers:
- User select area
- User click on "Edit" button next to one parking lot.

Preconditions:
- Mobile application is already launch.

- Administrator has been logged in

Post Conditions:

- **On Success:** Parking lot is updated by new value to server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select a car park	Show the car park detail information screen
2	User select a area	Show the area information screen
3	User click on "Edit" button next to once parking lot	Application navigate to edit parking lot screen
3	User fill in the textbox Select "Submit" button	New parking lot value is updated to server

Alternative Scenario:

- N/A

Exceptions:

- N/A

Business Rules:

- N/A

Use Case-14 specification			
Use-case no.	PGSS14	**Use-case version**	1.0
Use-case name	Delete Parking Lot		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:

- Administrator

Summary:

- This use case allow user to delete a parking lot on the system

Goal:

- Delete a parking lot on the server

Triggers:

- User select an area
- User click on "x" button next to the parking lot.

Preconditions:

- Mobile application is already launch.
- Administrator has been logged in

Post Conditions:

- **On Success:** Delete an area on server
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select once car park	Show the car park detail information on mobile
2	User select an area	Show the area information screen
3	User click on "Remove" or "x" button next to the parking lot	Application show the confirm button
4	User click "Accept"	Parking lot is removed from the server

Alternative Scenario:

- N/A

Exceptions:

- **N/A**

Business Rules:

- N/A

2.3.3. End User Use Case

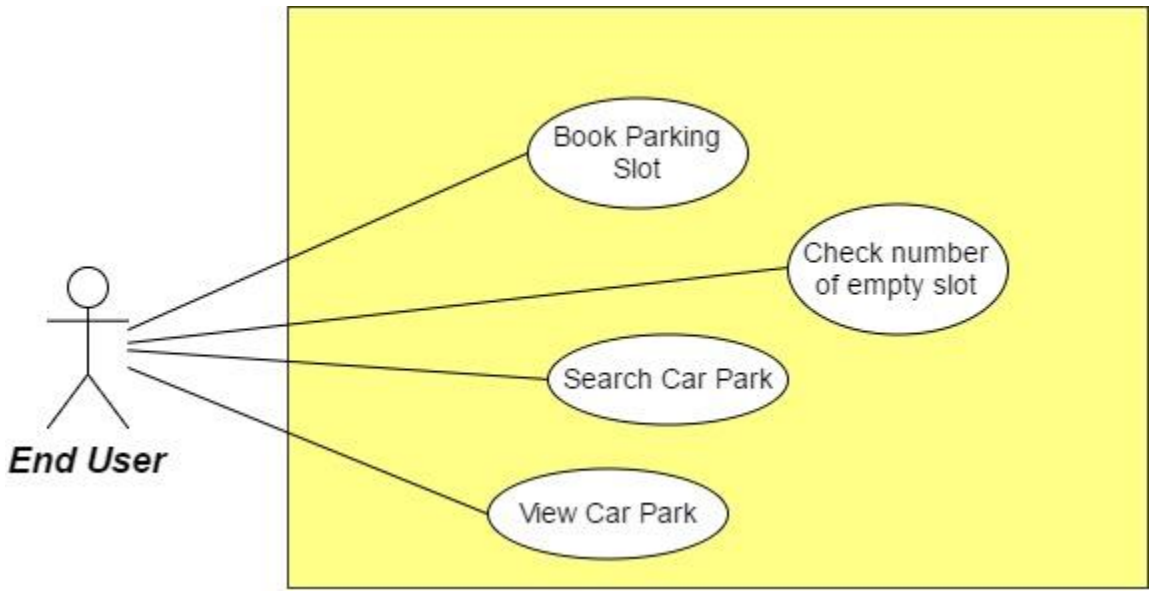


Figure 21: End User Use Case Diagram

Use Case-15 specification			
Use-case no.	PGSS15	**Use-case version**	1.0
Use-case name	Check number of empty slot		

Author	Bui Phu Hiep		
Date	13/02/17	Priority	High

Actor:
- End User

Summary:
- This use case allows user view number of empty slot in each car park

Goal:
- Show number of empty slot

Triggers:
- User login to the mobile application

Preconditions:
- Mobile application is already launch.
- End user had logged in.

Post Conditions:
- **On Success:** User know the number of empty slot in car park
- **On Failure:** Don't show number of empty slot in car park

Main Success Scenario:

No.	Actor Action	System Response
1	User log in to the application	Show the map with the marker as car park and the number, which indicate the number of empty slot

Alternative Scenario:
- N/A

Exceptions:
- The number will have tick/ exclamation points to show that the number is recently update or not.

Business Rules:
- Tick: recently update
- Exclamation points: number is not update in more than 1 hour.

Use Case-16 specification			
Use-case no.	PGSS16	Use-case version	1.0
Use-case name	Book parking slot		
Author	Bui Phu Hiep		
Date	13/02/17	Priority	High

Actor:
- End User

Summary:
- This use case allow user to book parking slot before go to the car park

Goal:
- Book the parking slot before go to car park

Triggers:

- User has selected the car park to book

Preconditions:

- Mobile application is already launch.
- End user had logged in.

Post Conditions:

- **On Success:** User book the parking slot success
- **On Failure:** Show error message when book

Main Success Scenario:

No.	Actor Action	System Response
1	User log in to the application	Show the map with the marker as car park and the number, which indicate the number of empty slot
2	User select the car park they want to book	Show the "Book" button if has empty slot
3	Fill information for transaction Click "Submit"	Make a transaction and set one parking slot to booked Show the address of booked parking slot to the user

Alternative Scenario:

- N/A

Exceptions:

- Transaction fail by 3rd party.

Business Rules:

- N/A

Use Case-17 specification			
Use-case no.	PGSS17	**Use-case version**	1.0
Use-case name	Search car park		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:

- End User

Summary:

- This use case allow user to search a car park by name or address

Goal:

- Show the searched car park

Triggers:

- User login to the mobile application

Preconditions:
- Mobile application is already launch.
- End user had logged in.

Post Conditions:
- **On Success:** Show the searched car park on the map if success
- **On Failure:** Show message error

Main Success Scenario:

No.	Actor Action	System Response
1	User log in to the application	Show the map with the marker as car park and the number, which indicate the number of empty slot
2	Enter the name or address in the search bar Press "Enter" or click "Search"	Find the car park base on name or address then focus on the map.

Alternative Scenario:

No.	Actor Action	System Response
1	User log in to the application	Show the map with the marker as car park and the number, which indicate the number of empty slot
2	Enter the name or address in the search bar Press "Enter" or click "Search"	Show message don't have car park if the name or address is incorrect

Exceptions:
- N/A

Business Rules:
- N/A

Use Case-18 specification			
Use-case no.	PGSS18	**Use-case version**	1.0
Use-case name	View Car Park		
Author	Bui Phu Hiep		
Date	13/02/17	**Priority**	High

Actor:
- End User

Summary:
- This use case allow user to view car park info.

Goal:
- User can view the information of car park.

Triggers:
- User select their car park.

Preconditions:
- Mobile application is already launch.
- Manager has been logged in

Post Conditions:
- **On Success:** User can view the information of car park.
- **On Failure:** Show error message

Main Success Scenario:

No.	Actor Action	System Response
1	User select once car park	Show the car park detail information on mobile

Alternative Scenario:
- N/A

Exceptions:
- N/A

Business Rules:
- N/A

3. Software System Attribute

3.1. Usability

- User controls all system components via only mobile application.
- The system can install easily.
- User can learn how to use the system fast.

3.2. Reliability

3.3. Availability

- The mechanical component requires electrical system to work well.
- Hardware components are easy to find in the market.

3.4. Security

- Mobile application requires authentication and authorization implement well because manager and end user use the same application.

3.5. Maintainability

- Use plug and play component so we can easily replace it.

3.6. Portability

- Easy to construct.

3.7. Performance

- Detection car is fast, less then 50ms.
- The speed of server can scale base on the budget easily.

4. Conceptual Diagram

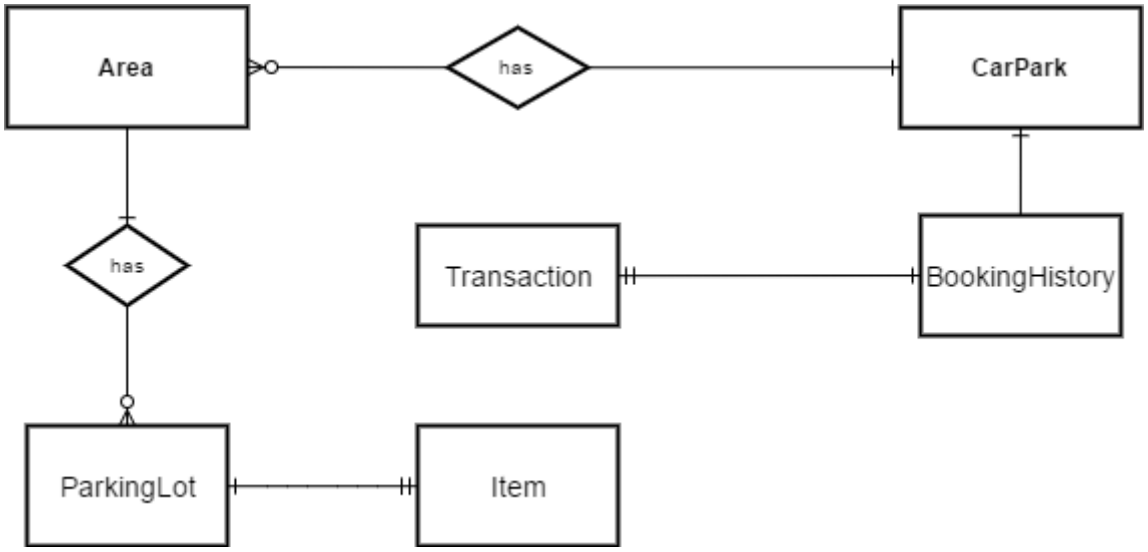


Figure 22: Conceptual Diagram

Data Dictionary

Entity Data dictionary: describe content of all entities	
Entity Name	**Description**
CarPark	Descript all car park information in the system
Area	Describe all area detail in car park
ParkingLot	Describe parking lot information in the area
Item	Describe hardware item in each parking lot
BookingHistory	Describe the booking history of the user
Transaction	Save the transaction of each booking

Table 17: Conceptual diagram data dictionary

D. Software – Hardware Design Description

1. Design Overview

This document describes the technical and user interface design of **PGSS.** It includes the architectural design, the detailed design of common functions and business functions and the design of database model.

The architectural design describes the overall architecture of the system and the architecture of each main component and subsystem.

The detailed design describes static and dynamic structure for each component and functions. It includes class diagrams, class explanations and sequence diagrams for each use case.

The database design describes the relationships between entities and details of each entity.

Document overview:

- Section 2: gives an overall description of the system architecture design.
- Section 3: gives component diagrams that describe the connection and integration of the system.
- Section 4: gives the detail design description which includes class diagram, class explanation, and sequence diagram to details the application functions.
- Section 5: describe screen design.
- Section 6: describe a fully attributed ERD.
- Section 7: describe algorithms.

2. System Architectural Design

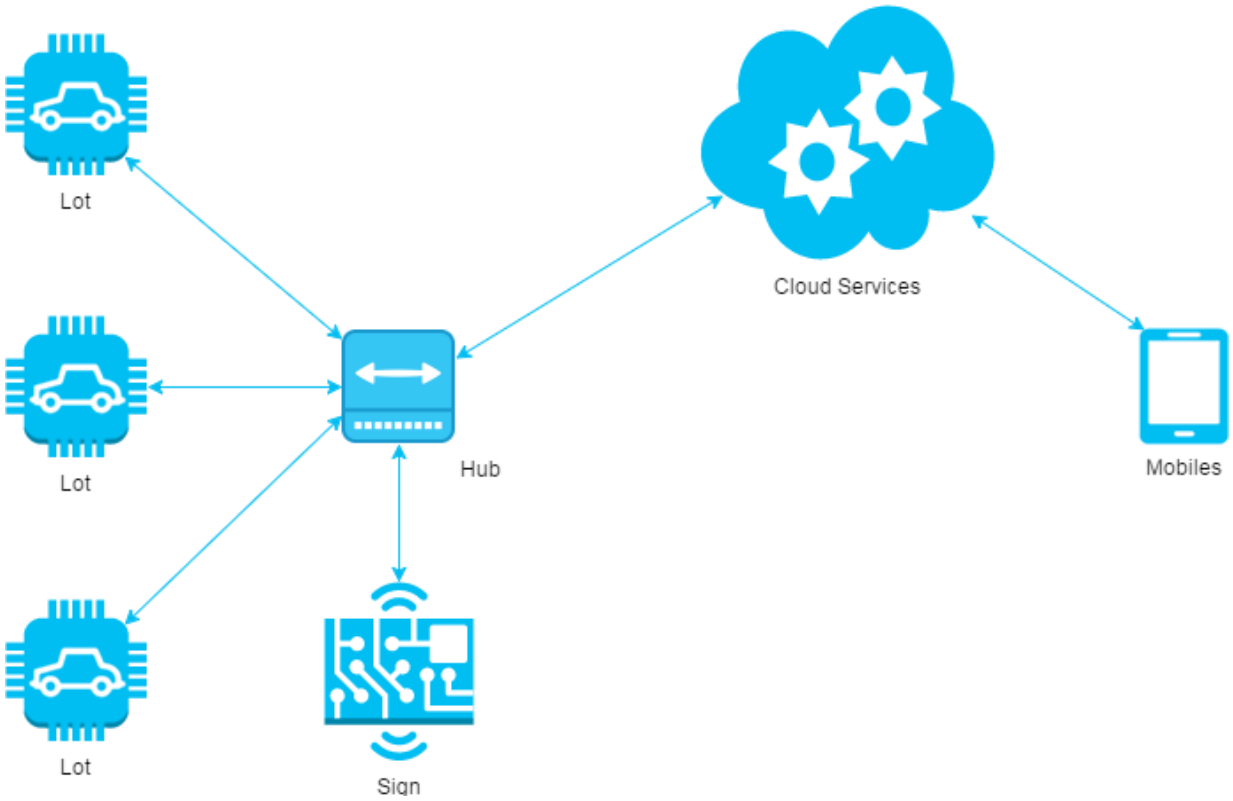


Figure 23: System architecture design

This diagram is referenced and modified from an original concept from: The Internet of Things: An Overview by The Internet Society (ISOC). In this device-to-gateway model, or more typically, the device-to-application-layer gateway (ALG) model, the IoT device connects through an ALG service as a conduit to reach a cloud service. The hub serves as a local gateway between individual IoT devices and a cloud service, but they can also bridge the interoperability gap between devices themselves.

2.1. Hardware Program Architecture Description

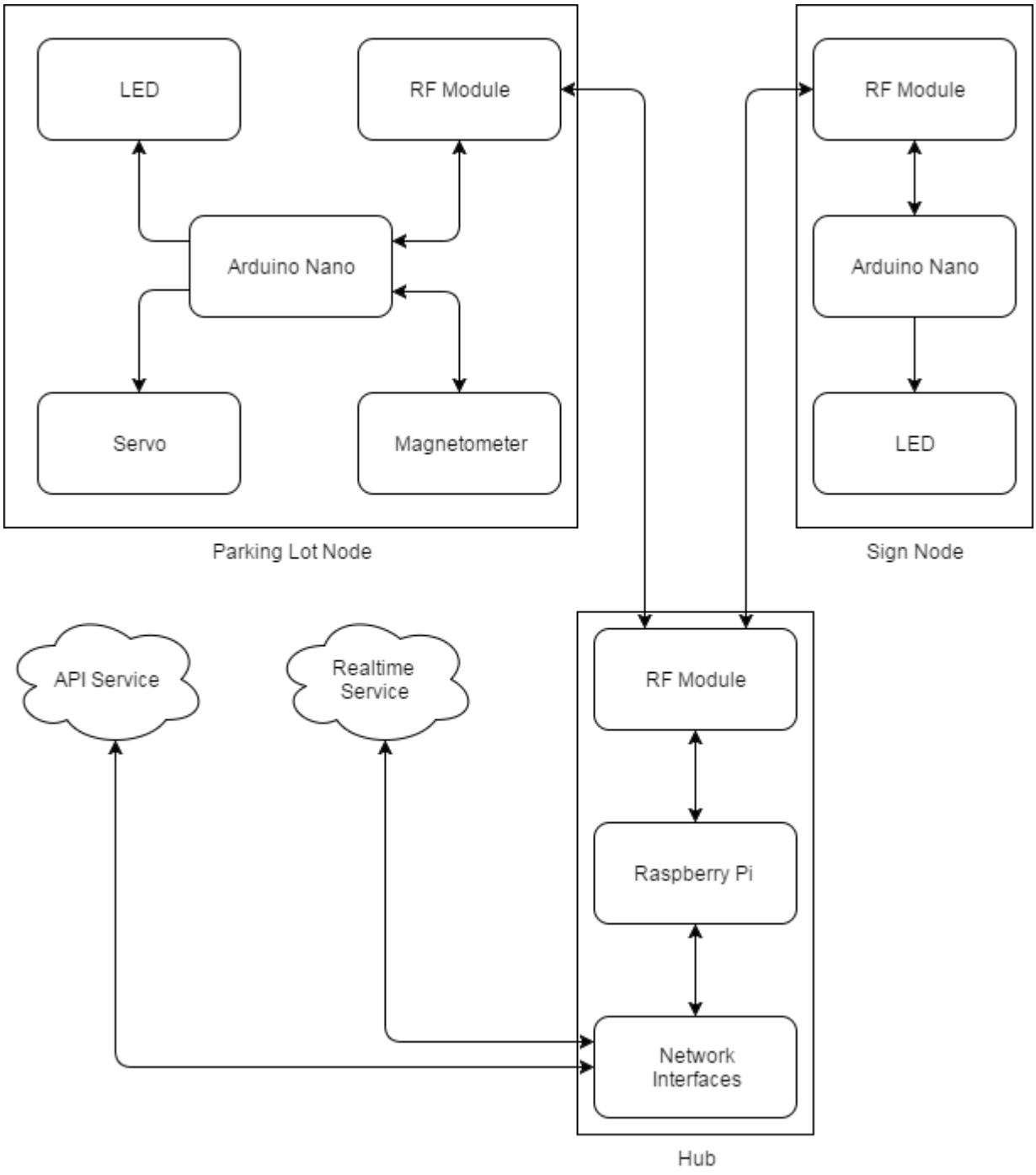


Figure 24: Hardware Program architectural

Parking Lot Node is a device placed at each lot in the car park. It plays the role as a passive IoT node and controls other components, which connects to it through several kinds of port to perform various works:

Arduino Nano: plays the roles as central processing unit for IoT node.

Magnetometer: an instrument that measures magnetism, which is used as metal detectors to detect ferrous metals in a car.

Servo: a linear actuator that allows for precise control of angular or linear position, velocity and acceleration. This will be used to control the barrier in each lot.

LED: a RGB LED used as an Indicator LED in each lot.

RF Module: a small device used to transmit/receive radio signals between two devices. This is used to create a network bridge for the whole system.

Sign Node is a device placed at each area and at the entrance of the car park. It also a passive node and used to display the available lots in each area:

Arduino Nano: plays the roles as central processing unit for IoT node.

RF Module: a small device used to transmit/receive radio signals between two devices. This is used to create a network bridge for the whole system.

LED: a large LED display screen

Hub is a stand-alone gateway device that has RF Module installed to communicate with all other IoT nodes. It also connects to cloud service, allowing the user to gain access to the devices using a smartphone app and an Internet connection:

Raspberry Pi: plays the roles as central processing unit for Hub node.

RF Module: a small device used to transmit/receive radio signals between two devices. This is used to create a network bridge for the whole system.

Network Interfaces: an interfaces to connect to the Internet.

API Service: is a cloud service that provides all necessary APIs and database.

Real time Service: a cloud service that provides real time communication between smartphone app and Hub.

2.2. Mobile Application Architecture Description

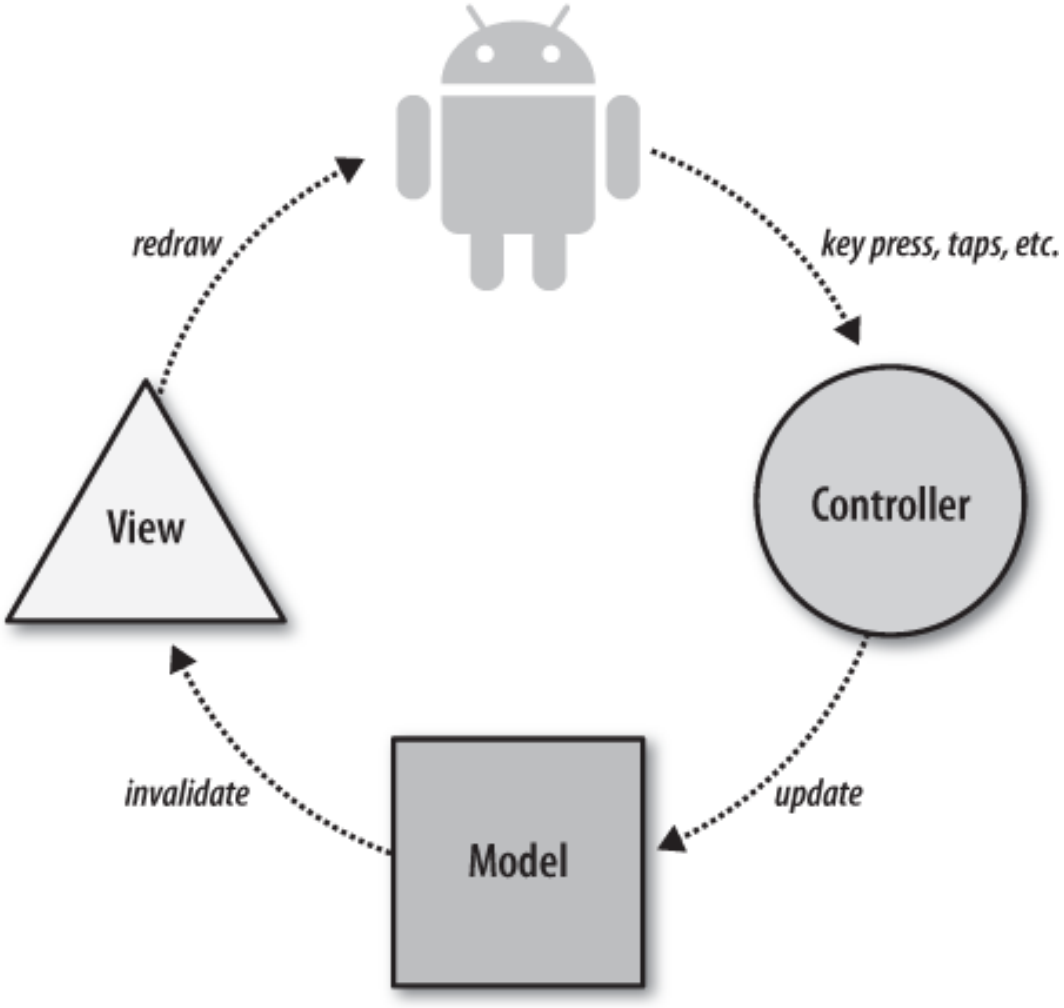


Figure 25: Mobile Application architecture

In Mobile Application, the system is developed under Standard Android Architecture. This is the default approach with layout files, Activities/Fragments acting as the controller and Models used for data and persistence. With this approach, Activities are in charge of processing the data and updating the views. Activities act like a controller in MVC but with some extra responsibilities that should be part of the view.

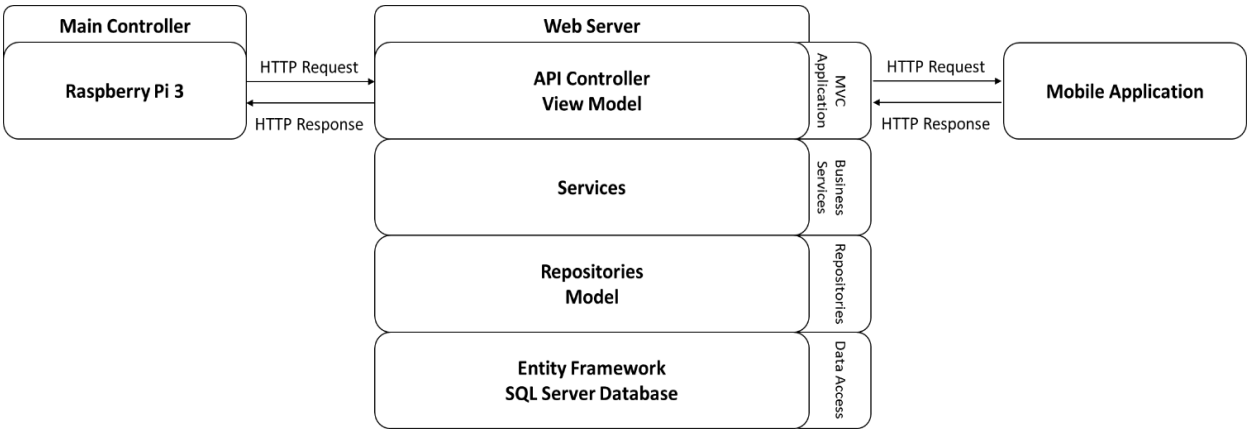
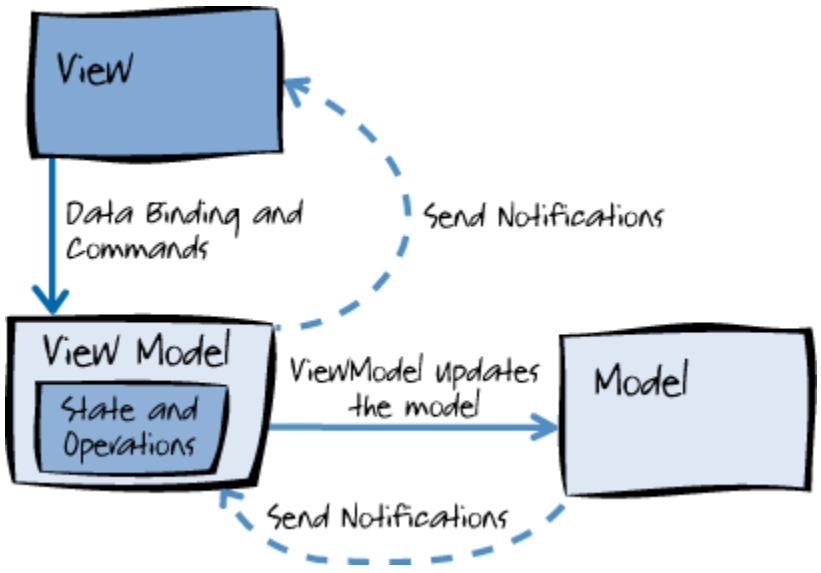


Figure 26: Hardware - Software Connection Architecture Description

2.3. Web API Architecture Description

In Web API, the server is design under MVVM Patern

There are three core components in the MVVM pattern: the model, the view, and the view model. Each serves a distinct and separate role. The following illustration shows the relationships between the three components.



3. Component Diagram

3.1. General Component Diagram

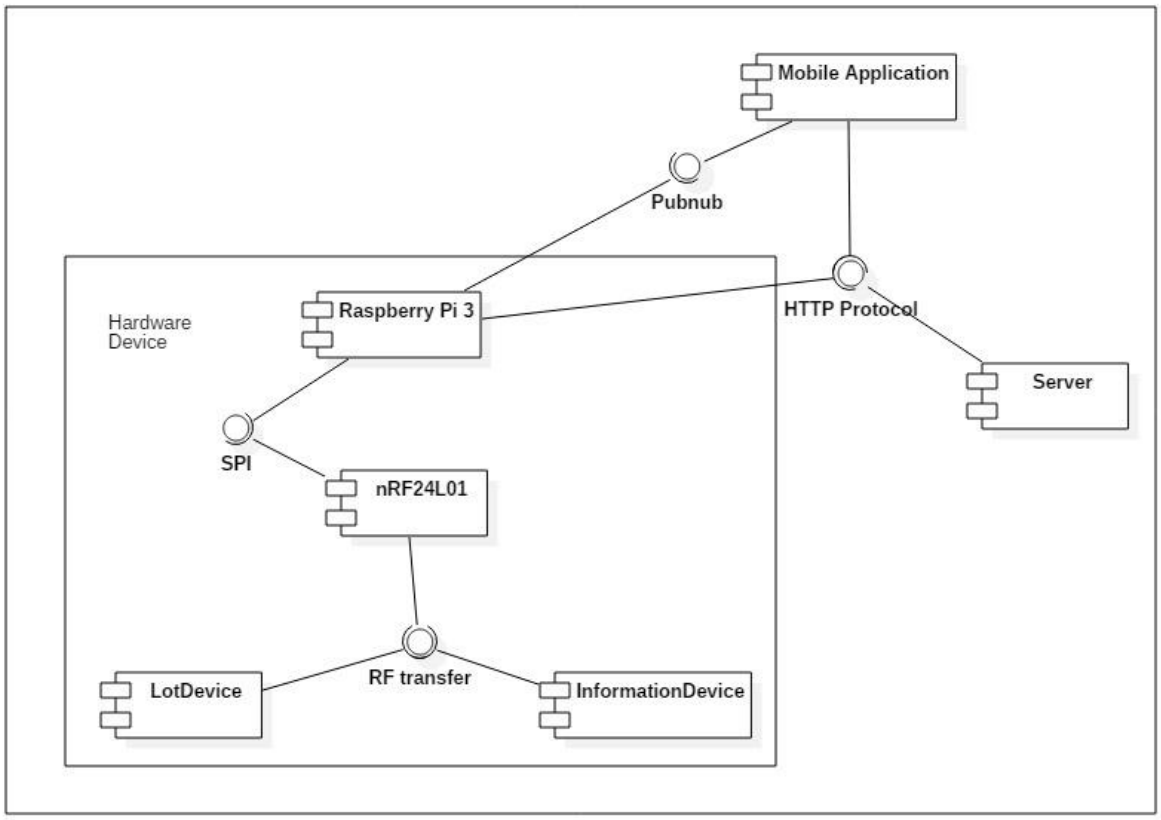


Figure 27: General Component Diagram

3.2. Lot Device Diagram

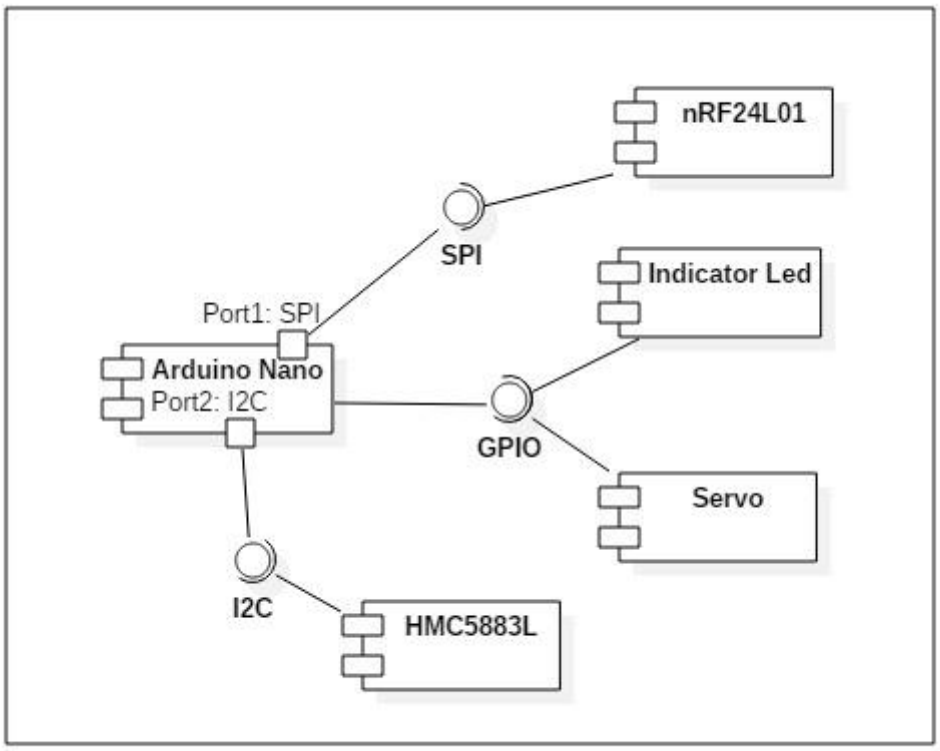


Figure 28: Lot Device Diagram

3.3. Information Device Diagram

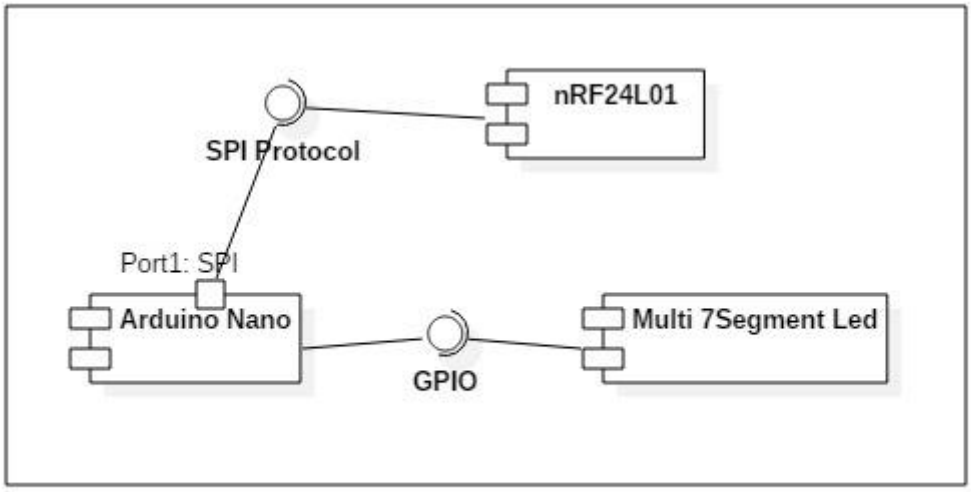


Figure 29: Information Device Diagram

3.4. Server Diagram

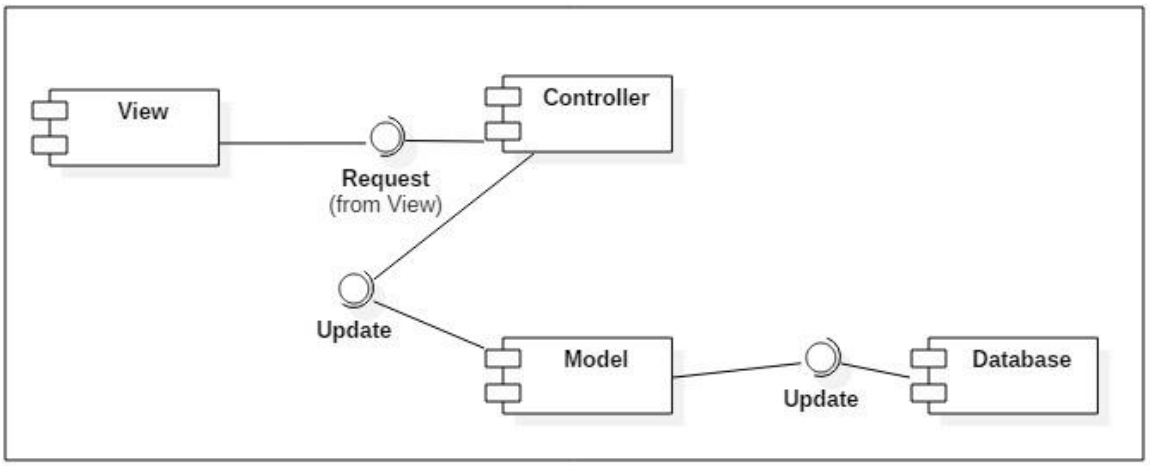


Figure 30: Server Diagram

3.5. Component Dictionary

Component Dictionary: Describes components	
Mobile Application	The application for user on android devices
Server	The Web API server, which make the interaction between the Raspberry Pi and the Mobile Application. It also save information to the database.
Raspberry Pi 3	The main controller in the hardware system that control all Lot Device and Information Device in each car park.
Lot Device	The device in each parking lot that control the indicator led, servo and use the magnetometer sensor to detect car.
Information Device	The number led board that show the number of empty lot in each area and number of empty lot in car park.
nRF24L01	The RF module, which help main controller, lot device and information device interact with each other.
Arduino Nano	The controller in lot device and information device.

Indicator Led	The RGB led to show the status of parking lot (empty, in used, reserved, disabled).
Servo	The barrier to prevent other car get in the parking lot when it is reserved.
HMC5883L	The magnetometer sensor to realize if the metal is near.
Pubnub	The third party API that help real time interaction.
HTTP Protocol	A transfer protocol to interact between server, mobile application and Raspberry Pi 3.
SPI	Synchronous serial communication interface used for short distance communication.
I²C	The inter-integrated circuit to communicate between arduino and magnetometer sensor.
GPIO	The communication between Arduino and led, servo, …
Multi 7-Segment Led	The 7 segment led to indicate number.
View	The JSON result of the API controller.
Controller	The API controller in the server.
Model	The entity model.
Database	The collection of data.

Table 18: Component dictionary

4. Detailed Description

4.1. Web API Detailed Description

4.1.1. Class Diagram

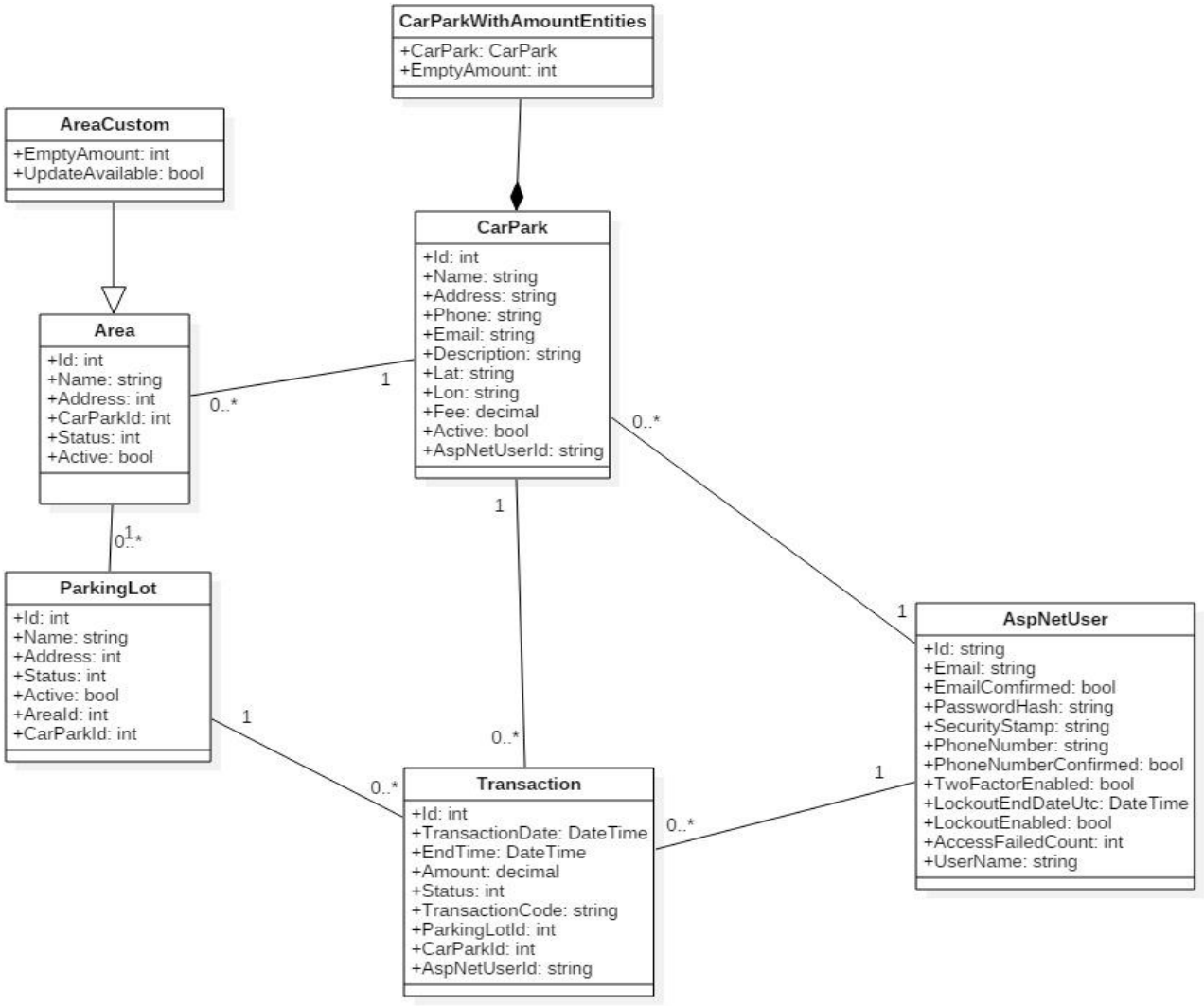


Figure 31: Model Class Diagram

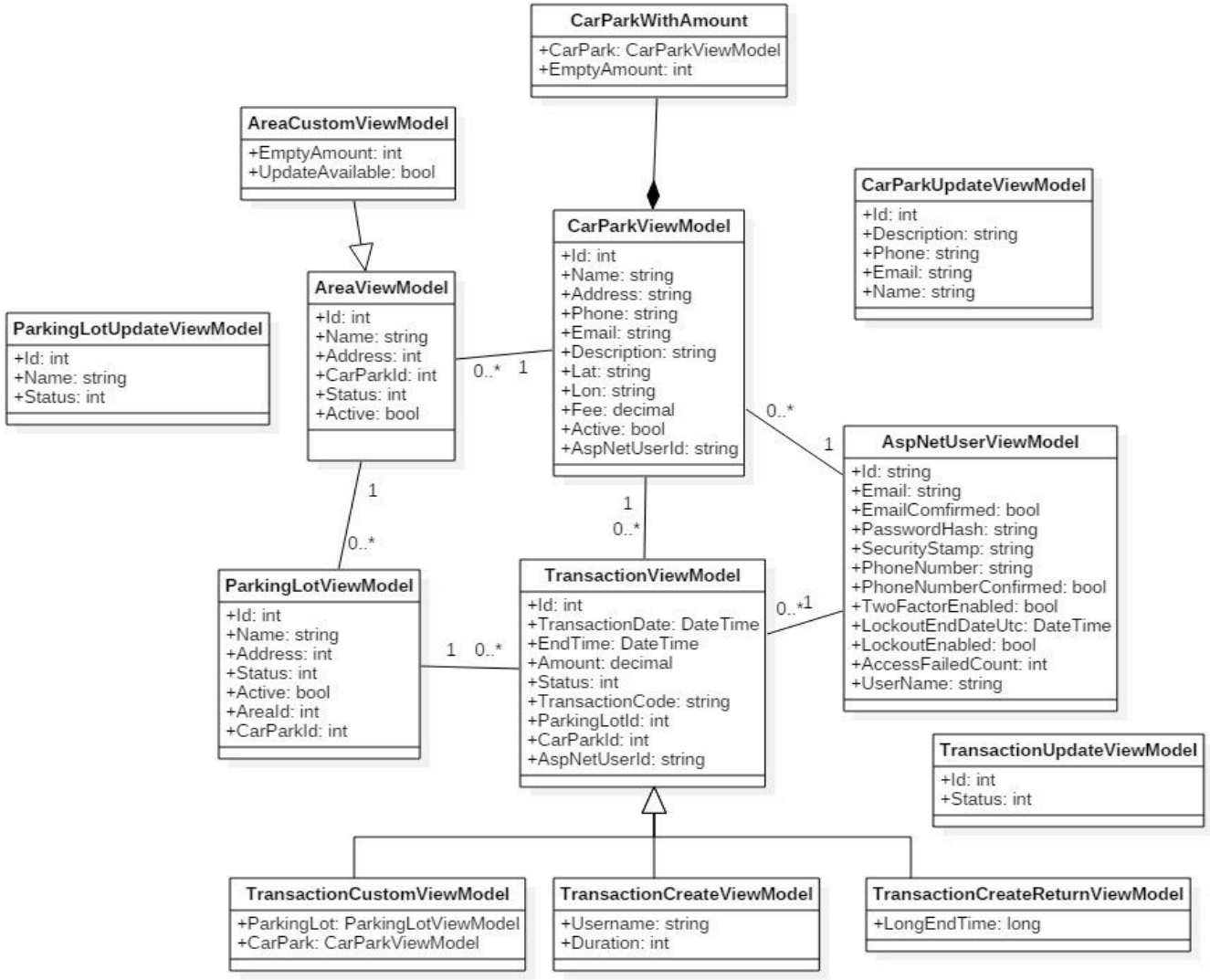


Figure 32: View Model Class Diagram

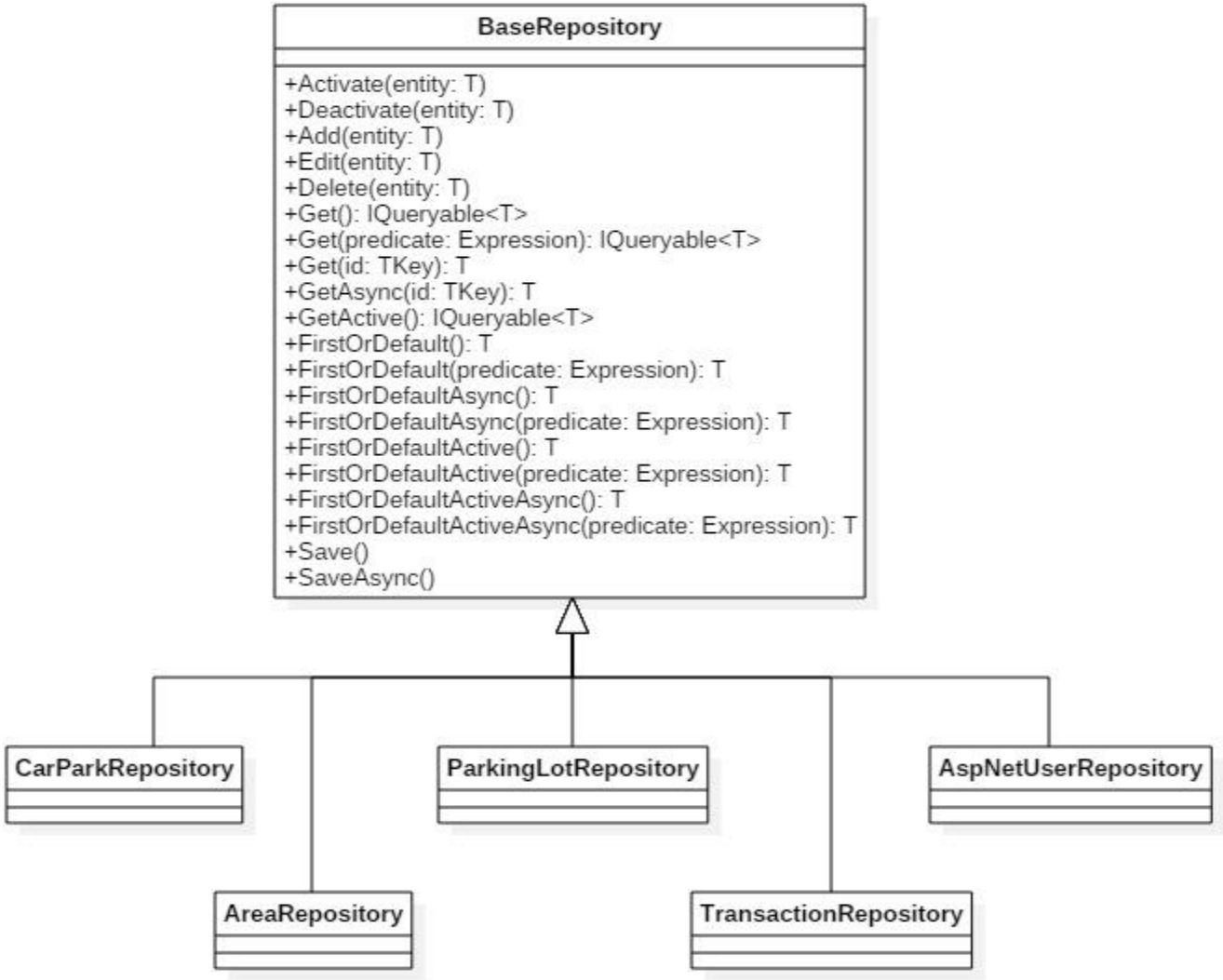


Figure 33: Repository Class Diagram

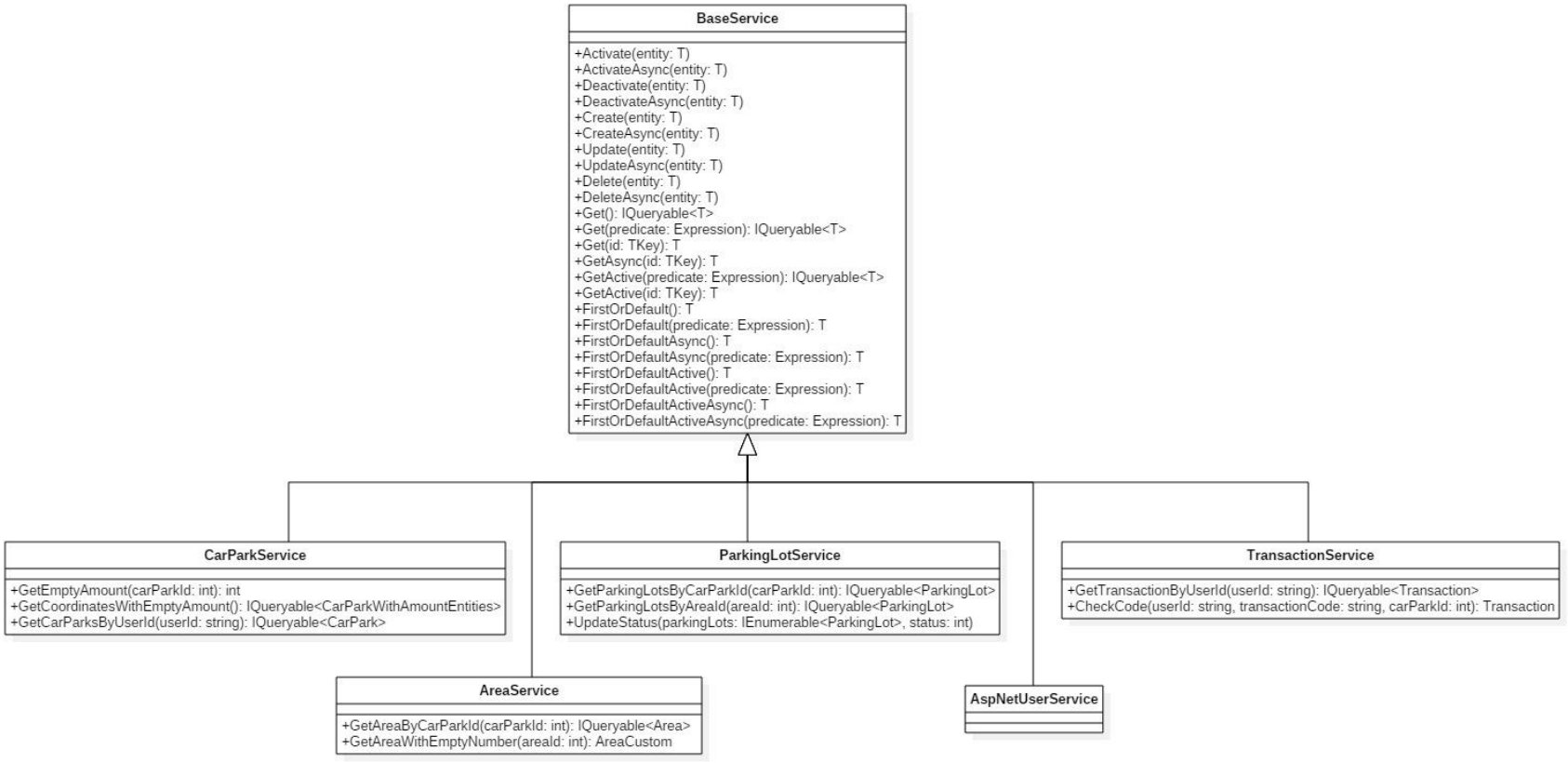


Figure 34: Service Class Diagram

4.1.2. Class Diagram Explanation

4.1.2.1. Model Class Diagram

CarPark

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each car park
Name	string	public	Name of car park
Address	string	public	Address of carpark
Phone	string	public	Phone number of car park
Email	string	public	Email of car park's owner
Description	string	public	Information of car park
Lat	string	public	Latitude of a car park
Lon	string	public	Longitude of a car park
Fee	decimal	public	Amount of money each hour
Active	bool	public	Value show that the car park is deleted or not
AspNetUserId	string	public	Id of the owner

Table 19: CarPark class attributes

CarParkWithAmountEntities

Attribute	Type	Visibility	Description
CarPark	CarPark	public	The car park entity
EmptyAmount	int	public	The number of empty lot in car park

Table 20: CarParkWithAmountEntities class attributes

Area

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each area
Name	string	public	Name of area
Address	int	public	Bit address of a area
CarParkId	int	public	Id of car park which hold area
Status	int	public	The number indicate the status of area
Active	bit	public	Value show that the area is deleted or not

Table 21: Area class attributes

AreaCustom

Attribute	Type	Visibility	Description
EmptyAmount	int	public	The number of empty lot in area
UpdateAvailable	bool	public	The boolean value check that area is available for update

Table 22: AreaCustom class attributes

ParkingLot

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each parking lot
Name	string	public	Name of parking lot
Address	int	public	Bit address of a parking lot
Status	int	public	The number indicate the status of parking lot
Active	bit	public	Value show that the parking lot is deleted or not
AreaId	int	public	Id of area which hold parking lot
CarParkId	int	public	Id of car park which hold parking lot

Table 23: ParkingLot class attributes

Transaction

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each transaction
TransactionDate	string	public	The time customer book the parking lot
EndTime	int	public	End of the booking time
Amount	int	public	The total fee of transaction
Status	bit	public	The number indicate status of the transaction
TransactionCode	string	public	The code for the user to enter in the car park to unlock the booked parking lot
ParkingLotId	int	public	Id of the reserved lot
CarParkId	int	public	Id of car park which hold parking lot
AspNetUserId	string	public	Id of the user who reserved

Table 24: Transaction class attributes

AspNetUser

Attribute	Type	Visibility	Description
Id	string	public	Unique Id for each user
Email	string	public	Email of the user
EmailConfirm	bool	public	Check if the user confirm the email
PasswordHash	string	public	Hash of the user password, don't save exact password
SecurityStamp	string	public	
PhoneNumber	string	public	The number of the user
PhoneNumberConfirmed	bool	public	Check if the number is confirmed
TwoFactorEnabled	bool	public	
LockoutEndDateUtc	DateTime	public	
LockoutEnabled	bool	public	
AccessFailedCount	int	public	The number of time access failed
UserName	string	public	Username of the user

Table 25: AspNetUser class attributes

4.1.2.2. View Model Class Diagram

CarParkViewModel

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each car park
Name	string	public	Name of car park
Address	string	public	Address of carpark
Phone	string	public	Phone number of car park
Email	string	public	Email of car park's owner
Description	string	public	Information of car park
Lat	string	public	Latitude of a car park
Lon	string	public	Longitude of a car park
Fee	decimal	public	Amount of money each hour
Active	bool	public	Value show that the car park is deleted or not
AspNetUserId	string	public	Id of the owner

Table 26: CarParkViewModel class attributes

CarParkWithAmount

Attribute	Type	Visibility	Description
CarPark	CarParkViewModel	public	The car park entity
EmptyAmount	int	public	The number of empty lot in car park

Table 27: CarParkWithAmount class attributes

CarParkUpdateViewModel

Attribute	Type	Visibility	Description
Id	int	public	Id of update car park
Description	string	public	New description
Phone	string	public	New phone number
Email	string	public	New email
Name	string	public	New name

Table 28: CarParkUpdateViewModel class attributes

AreaViewModel

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each area
Name	string	public	Name of area
Address	int	public	Bit address of a area
CarParkId	int	public	Id of car park which hold area
Status	int	public	The number indicate the status of area
Active	bit	public	Value show that the area is deleted or not

Table 29: AreaViewModel class attributes

AreaCustomViewModel

Attribute	Type	Visibility	Description
EmptyAmount	int	public	The number of empty lot in area
UpdateAvailable	bool	public	The boolean value check that area is available for update

Table 30: AreaCustomViewModel class attributes

ParkingLotViewModel

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each parking lot

Name	string	public	Name of parking lot
Address	int	public	Bit address of a parking lot
Status	int	public	The number indicate the status of parking lot
Active	bit	public	Value show that the parking lot is deleted or not
AreaId	int	public	Id of area which hold parking lot
CarParkId	int	public	Id of car park which hold parking lot

Table 31: ParkingLotViewModel class attributes

ParkingLotUpdateViewModel

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each parking lot
Name	string	public	New name if change
Status	int	public	New status if change

Table 32: ParkingLotUpdateViewModel class attributes

TransactionViewModel

Attribute	Type	Visibility	Description
Id	int	public	Unique Id for each transaction
TransactionDate	string	public	The time customer book the parking lot
EndTime	int	public	End of the booking time
Amount	int	public	The total fee of transaction
Status	bit	public	The number indicate status of the transaction
TransactionCode	string	public	The code for the user to enter in the car park to unlock the booked parking lot
ParkingLotId	int	public	Id of the reserved lot
CarParkId	int	public	Id of car park which hold parking lot
AspNetUserId	string	public	Id of the user who reserved

Table 33: Transaction class attributes

TransactionCustomViewModel

Attribute	Type	Visibility	Description
ParkingLot	ParkingLotViewModel	public	ParkingLot's model

CarPark	CarParkViewModel	public	CarPark, where transaction is booked

Table 34: TransactionCustomViewModel class attributes

TransactionCreateViewModel

Attribute	Type	Visibility	Description
Username	string	public	Username of the customer
Duration	int	public	Time of booking

Table 35: TransactionCreateViewModel class attributes

TransactionCreateReturnViewModel

Attribute	Type	Visibility	Description
LongEndTime	long	public	DateTime in number format

Table 36: TransactionCreateReturnViewModel class attributes

TransactionUpdateViewModel

Attribute	Type	Visibility	Description
Id	int	public	Id of the transaction
Status	int	public	The integer indicate the status of transaction

Table 37: TransactionUpdateViewModel class attributes

AspNetUser

Attribute	Type	Visibility	Description
Id	string	public	Unique Id for each user
Email	string	public	Email of the user
EmailConfirm	bool	public	Check if the user confirm the email
PasswordHash	string	public	Hash of the user password, don't save exact password
SecurityStamp	string	public	
PhoneNumber	string	public	The number of the user
PhoneNumberConfirmed	bool	public	Check if the number is confirmed
TwoFactorEnabled	bool	public	
LockoutEndDateUtc	DateTime	public	
LockoutEnabled	bool	public	
AccessFailedCount	int	public	The number of time access failed

UserName	string	public	Username of the user

Table 38: AspNetUser class attributes

4.1.2.3. Repository Class Diagram

BaseRepository

Method	Type	Visibility	Description
Activate	void	public	Set the active attribute of current model to True
Deactivate	void	public	Set the active attribute of current model to False
Add	void	public	Add a model to database
Edit	void	public	Edit a input model
Delete	void	public	Delete a input model in database
Get	IQueryable<T>	public	Get a list of model with condition in database
Get	T	public	Get a model by Id in database
GetAsync	T	public	Get a model by Id in database asynchronous
GetActive	IQueryable<T>	public	Get a list of model with condition in database asynchronous
FirstOrDefault	T	public	Get first model in a list with condition in database
FirstOrDefaultAsync	T	public	Get first model in a list with condition in database asynchronous
FirstOrDefaultActive	T	public	Get first model in a list of active item with condition in database
FirstOrDefaultActiveAsync	T	public	Get first model in a list of active item with condition in database asynchronous
Save	void	public	Save a current state of entity to database
SaveAsync	void	public	Save a current state of

			entity to database asynchronous

Table 39: BaseRepository class methods

4.1.2.4. Service Class Diagram

Base Service

Method	Type	Visibility	Description
Activate	void	public	Set the active attribute of current model to True
ActivateAsync	void	public	Set the active attribute of current model to True asynchronous
Deactivate	void	public	Set the active attribute of current model to False
DeactivateAsync			Set the active attribute of current model to False asynchronous
Create	void	public	Create a new item in database
CreateAsync	void	public	Create a new item in database asynchronous
Update	void	public	Update a item in database
UpdateAsync	void	public	Update an item in database asynchronous
Delete	void	public	Delete an item in database
DeleteAsync	void	public	Delete an item in database asynchronous
Get	IQueryable<T>	public	Get a list of model with condition in database
Get	T	public	Get a model by Id in database
GetAsync	T	public	Get a model by Id in database asynchronous
GetActive	IQueryable<T>	public	Get a list of model with condition in database asynchronous
GetActive	T	public	Get active model by Id in

			database if exist
FirstOrDefault	T	public	Get first model in a list with condition in database
FirstOrDefaultAsync	T	public	Get first model in a list with condition in database asynchronous
FirstOrDefaultActive	T	public	Get first model in a list of active item with condition in database
FirstOrDefaultActiveAsync	T	public	Get first model in a list of active item with condition in database asynchronous

Table 40: BaseService class methods

AreaService

Method	Type	Visibility	Description
GetAreaByCarParkId	IQueryable<Area>	public	Get the area with input car park id
GetAreaWithEmptyNumber	AreaCustom	public	Get the area with empty number of lot by area id

Table 41: AreaService class methods

CarParkService

Method	Type	Visibility	Description
GetEmptyAmount	int	public	Get the number of empty lot by car park id
GetCoordinates WithEmptyAmount	IQueryable <CarParkWith AmountEntities>	public	Get the list of car park with coordinate and number of empty lot
GetCarParksByUserId	IQueryable <CarPark>	public	Get the list of car park by AspNetUserId (owner of car park)

Table 42: CarParkService class methods

ParkingLotService

Method	Type	Visibility	Description
GetParkingLotsByCarParkId	IQueryable<ParkingLot>	public	Get the list of parking lot in the

			car park
GetParkingLotsByAreaId	IQueryable<ParkingLot>	public	Get the list of parking lot in the area
UpdateStatus	void	public	Change the status of a list parking lot

Table 43: ParkingLotService class methods

Transaction Service

Method	Type	Visibility	Description
GetTransactionByUserId	IQueryable<Transaction>	public	Get the list of transaction create by a user
CheckCode	Transaction	public	Get the transaction bases on user, transaction code and car park

Table 44: TransactionService class methods

4.1.3. Interaction Diagram

4.1.3.1. Register

Summary: This diagram shows the process to create a new user.

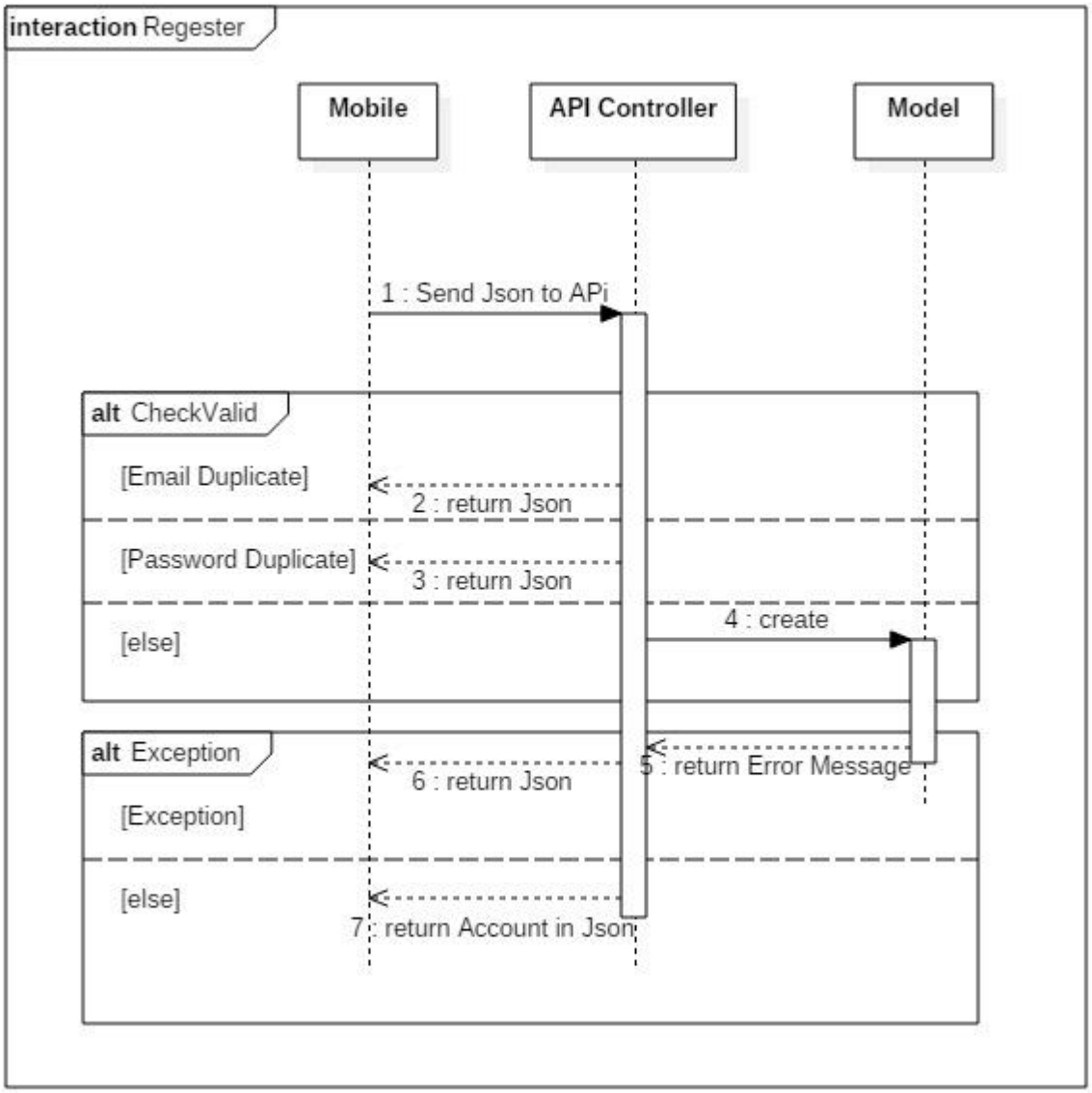


Figure 35: Register Diagram

4.1.3.2. Login

Summary: This diagram shows the process to authenticate the user on the server when they login on mobile

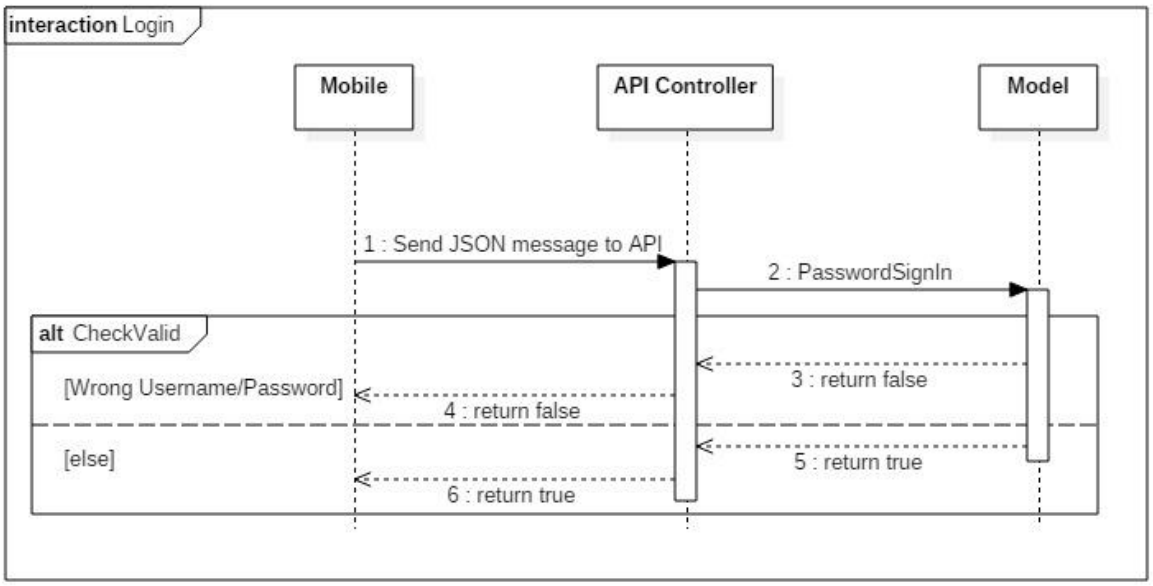


Figure 36: Login Diagram

4.1.3.3. Get Car Park

Summary: This diagram shows the process to get the list of available car park and return to the mobile application.

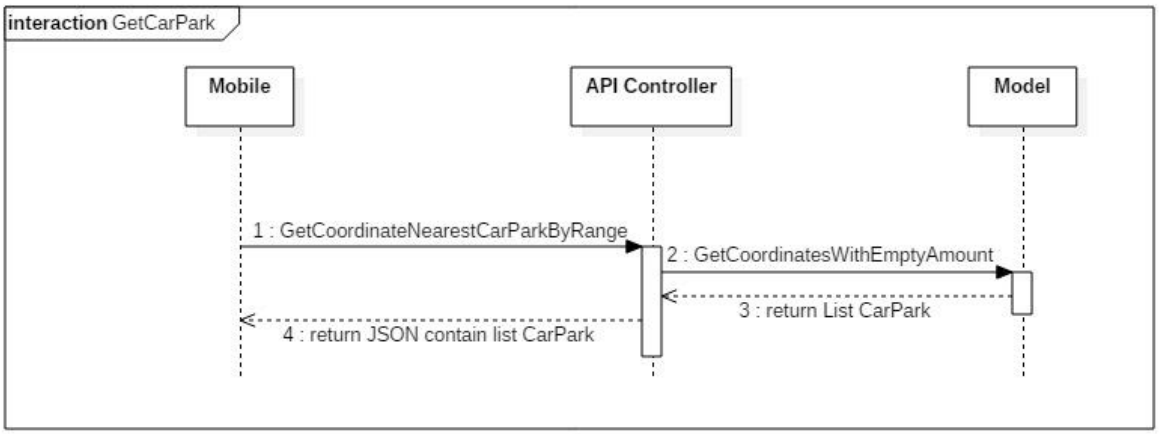


Figure 37: GetCarPark Diagram

4.1.3.4. Get Area

Summary: This diagram shows the process to get the list of available area in selected car park

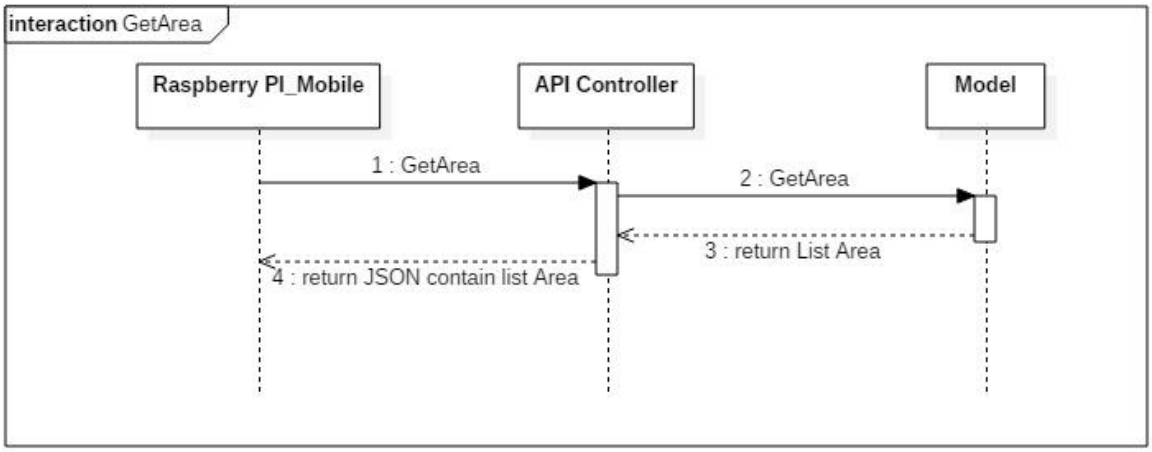


Figure 38: GetArea Diagram

4.1.3.5. Get Parking Lot

Summary: This diagram shows the process to get the list of available parking lot in selected car park

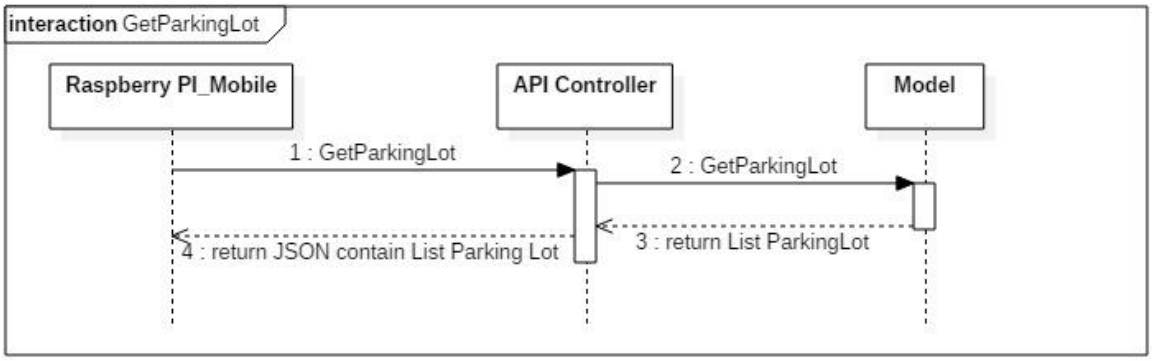


Figure 39: GetParkingLot Diagram

4.1.3.6. Update Area

Summary: This diagram shows the process to update the status/name of an area. If area has the lot in use, it will not be updated.

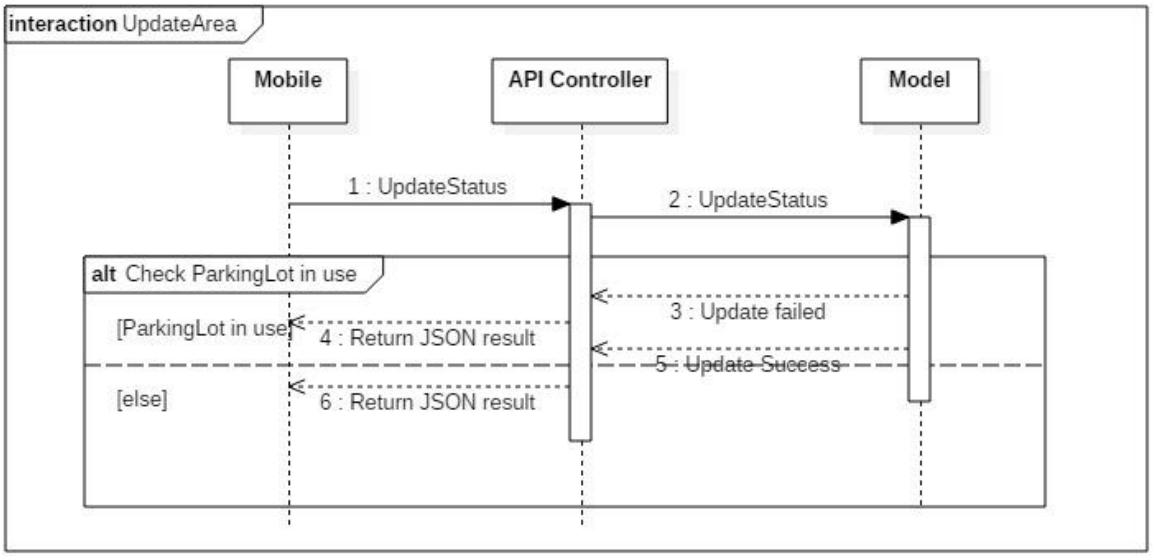


Figure 40: UpdateArea Diagram

4.1.3.7. Update Parking Lot

Summary: This diagram shows the process to update the status/name of a parking lot. If selected parking lot is in use, it will not be updated.

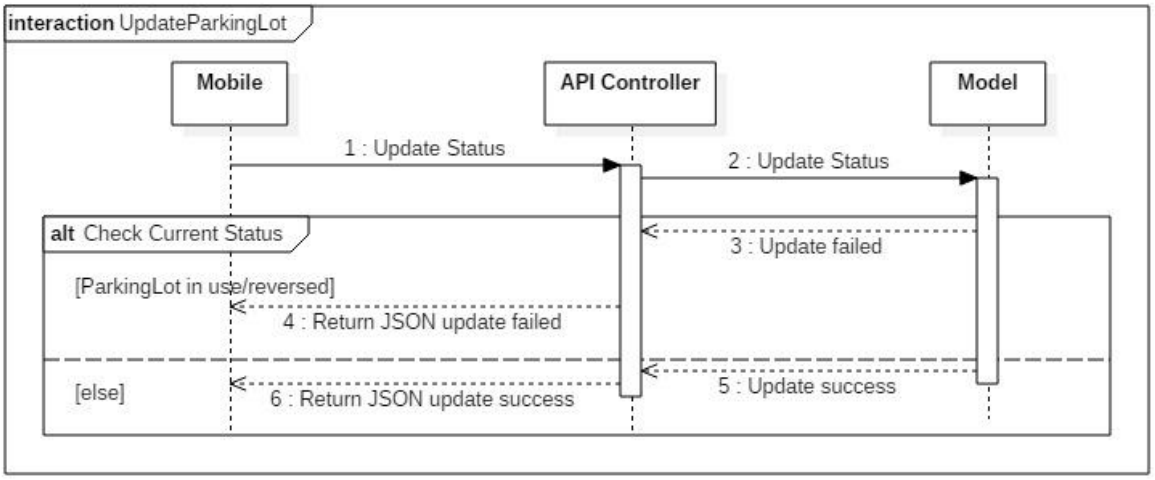


Figure 41: UpdateParkingLot Diagram

4.1.3.8. Update List Status Parking Lot

Summary: This diagram shows the process to update the status/name of a list of area. If once parking lot is in used, it will not be update.

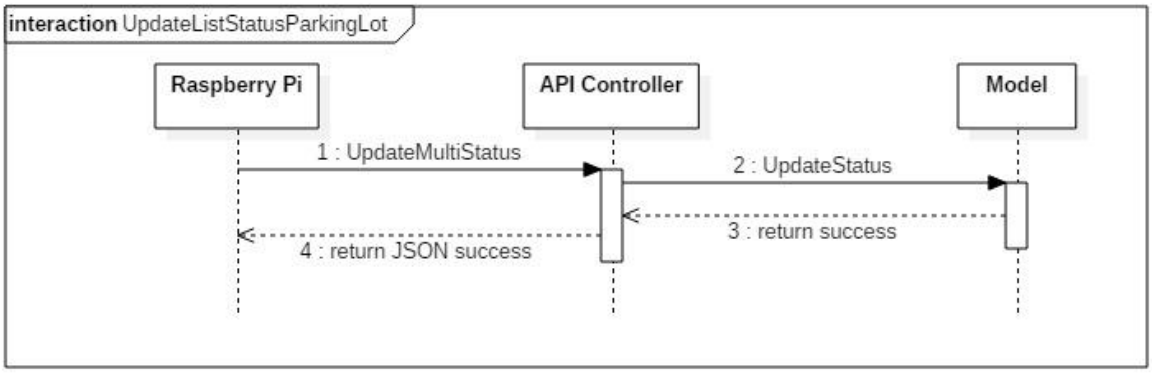


Figure 42: UpdateListStatusParkingLot Diagram

4.2. Mobile Detailed Description

4.2.1. Class Diagram

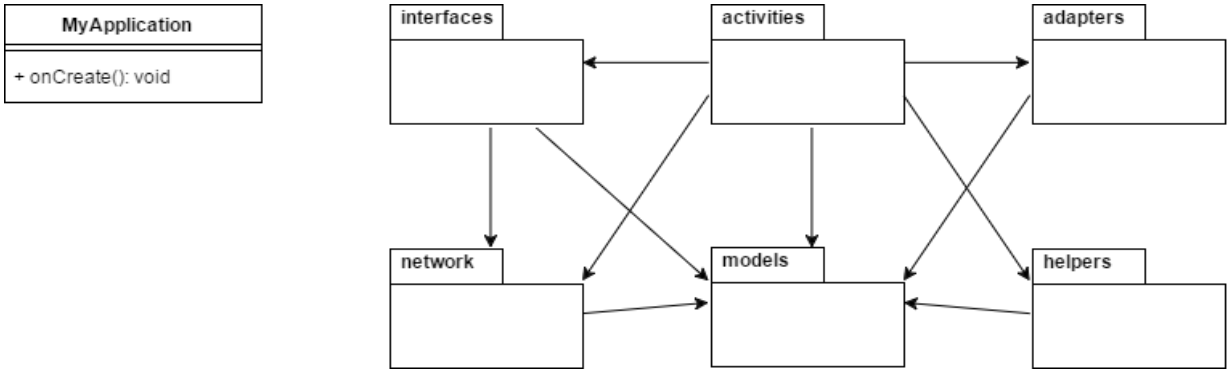


Figure 43: Main class diagram and package view

Packages Dictionary: describe packages	
Package Name	**Description**
Interfaces	Package that provides interfaces for API communication
Activities	Package that provides all activities for Mobile Application
Adapters	Package that provides adapters for Recycler View / List View / Info Window
Network	Package that provides network models and network classes
Models	Package that provides all models
Helpers	Package that provides all helper class for database, network, pubnub…

Table 45: Packages dictionary

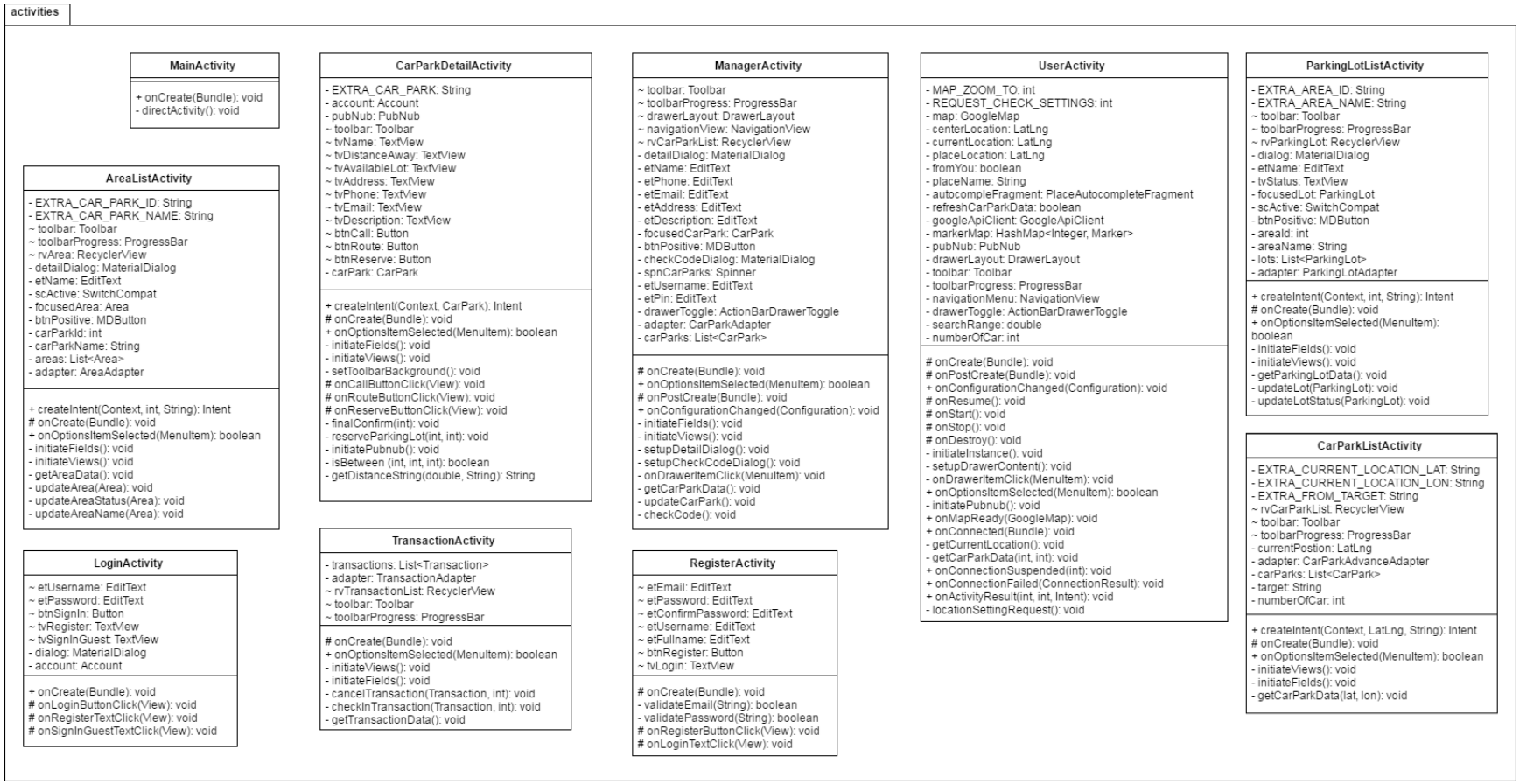


Figure 44: Activities package class diagram

Class Dictionary: describes class	
Class Name	**Description**
MainActivity	Activity to handle direction of activity flow when app start
CarParkDetailActivity	Activity to provide detail information of car park and to handle request from user
ManagerActivity	Activity to provide manager with car parks information and to handle commands
AreaListActivity	Activity to provide list of areas in car park and handle commands
ParkingLotListActivity	Activity to provide list of parking lots in area and handle commands
LoginActivity	Activity to handle login request of user
RegisterActivity	Activity to handle register request of user
UserActivity	Activity to provide user with car parks information in map view and handle commands and directions to other activities
CarParkListActivity	Activity to provide user with car parks information in list view and handle commands
TransactionActivity	Activity to provide user with list of history transactions and handle commands

Table 46: Activities package class dictionary

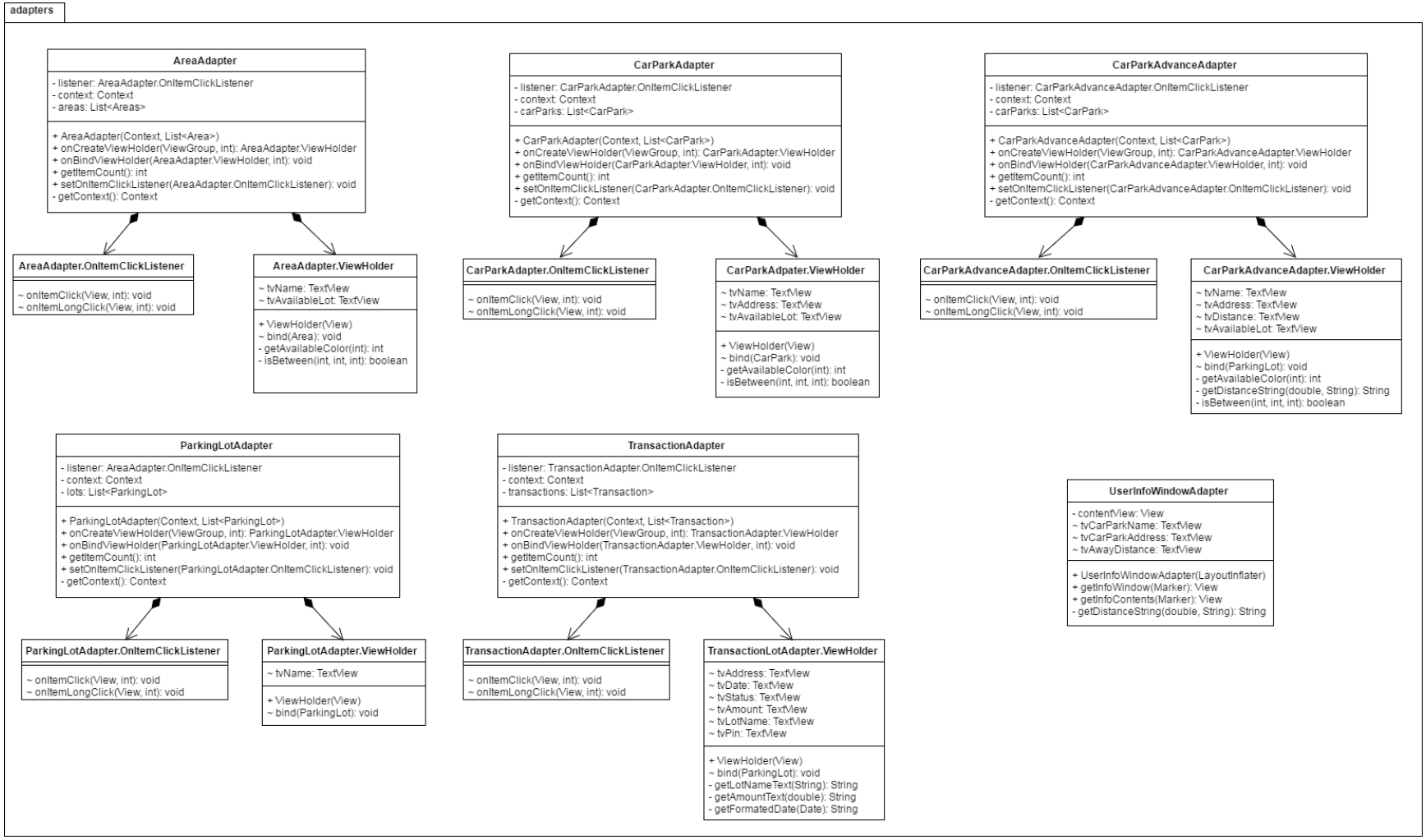


Figure 45: Adapters package class diagram

Class Dictionary: describes class	
Class Name	**Description**
AreaAdapter	Provides a binding from Area data set to views that are displayed within a Recycler View
AreaAdapter .OnItemClickListener	Interface definition for a callback to be invoked when an item in AreaAdapter has been click
AreaAdapter .ViewHolder	Describes an item view and metadata about its place within the Recycler View
CarParkAdapter	Provides a binding from Car Park data set to views that are displayed within a Recycler View
CarParkAdapter .OnItemClickListener	Interface definition for a callback to be invoked when an item in CarParkAdapter has been click
CarParkAdapter .ViewHolder	Describes an item view and metadata about its place within the Recycler View
ParkingLotAdapter	Provides a binding from Parking Lot data set to views that are displayed within a Recycler View
ParkingLotAdapter .OnItemClickListener	Interface definition for a callback to be invoked when an item in ParkingLotdapter has been click
ParkingLotAdapter .ViewHolder	Describes an item view and metadata about its place within the Recycler View
CarParkAdvanceAdapter	Provides a binding from Car Park Advance data set to views that are displayed within a Recycler View
CarParkAdvanceAdapter .OnItemClickListener	Interface definition for a callback to be invoked when an item in CarParkAdvanceAdapter has been click
CarParkAdvanceAdapter .ViewHolder	Describes an item view and metadata about its place within the Recycler View
TransactionAdapter	Provides a binding from Transaction data set to views that are displayed within a Recycler View
TransactionAdapter .OnItemClickListener	Interface definition for a callback to be invoked when an item in TransactionAdapter has been click

TransactionAdapter .ViewHolder	Describes an item view and metadata about its place within the Recycler View
UserInfoWindowAdapter	Provides views for customized rendering of info windows

Table 47: Adapters package class dictionary

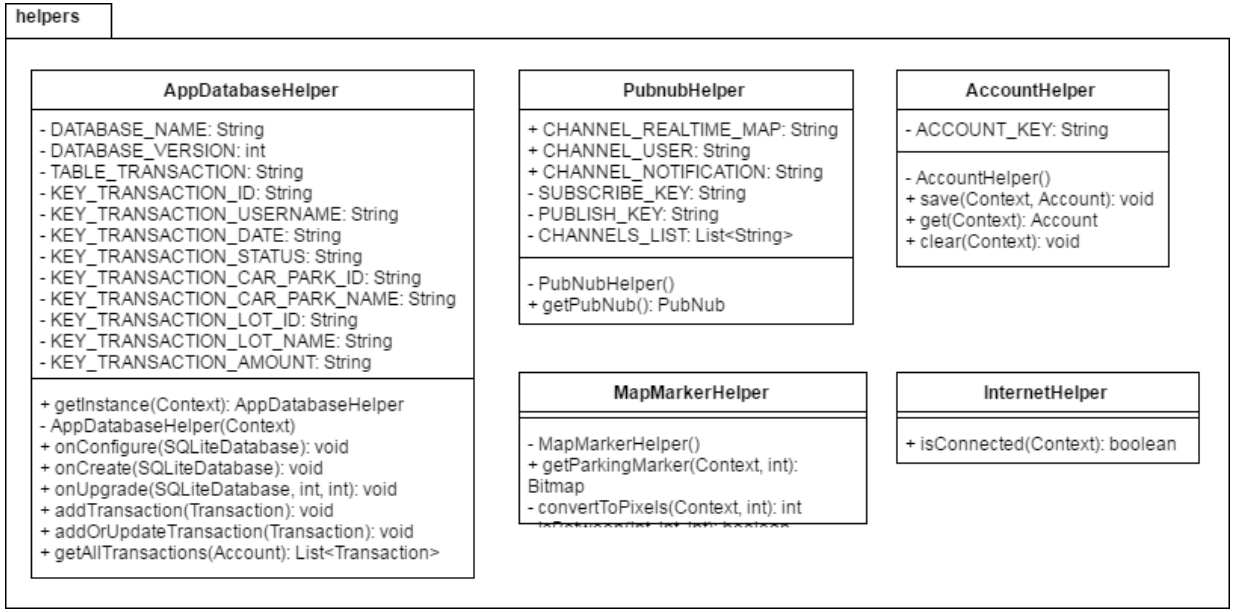


Figure 46: Helpers package class diagram

Class Dictionary: describes class	
Class Name	**Description**
AppDatabaseHelper	A helper class provides API to work with SQLite
PubnubHelper	A helper class provides API to work with PubNub
AccountHelper	A helper class provides API to work with account data in shared preferences
MapMarkerHelper	A helper class provides utility for Google Map Marker
InternetHelper	A helper class provides utility for network

Table 48: Helpers package class dictionary

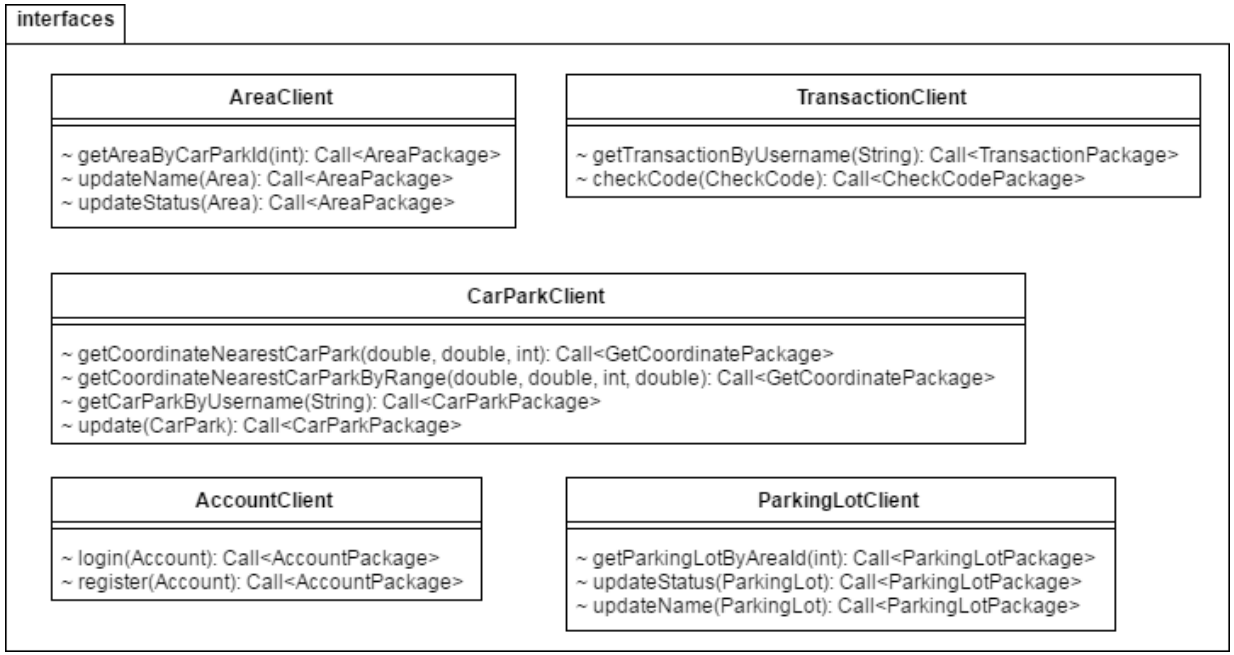


Figure 47: Interfaces package class diagram

Class Dictionary: describes class	
Class Name	**Description**
AreaClient	An interface to consume API call for Area
TransactionClient	An interface to consume API call for Transaction
CarParkClient	An interface to consume API call for Car Park
AccountClient	An interface to consume API call for Account
ParkingLotClient	An interface to consume API call for Parking Lot

Table 49: Interfaces package class dictionary

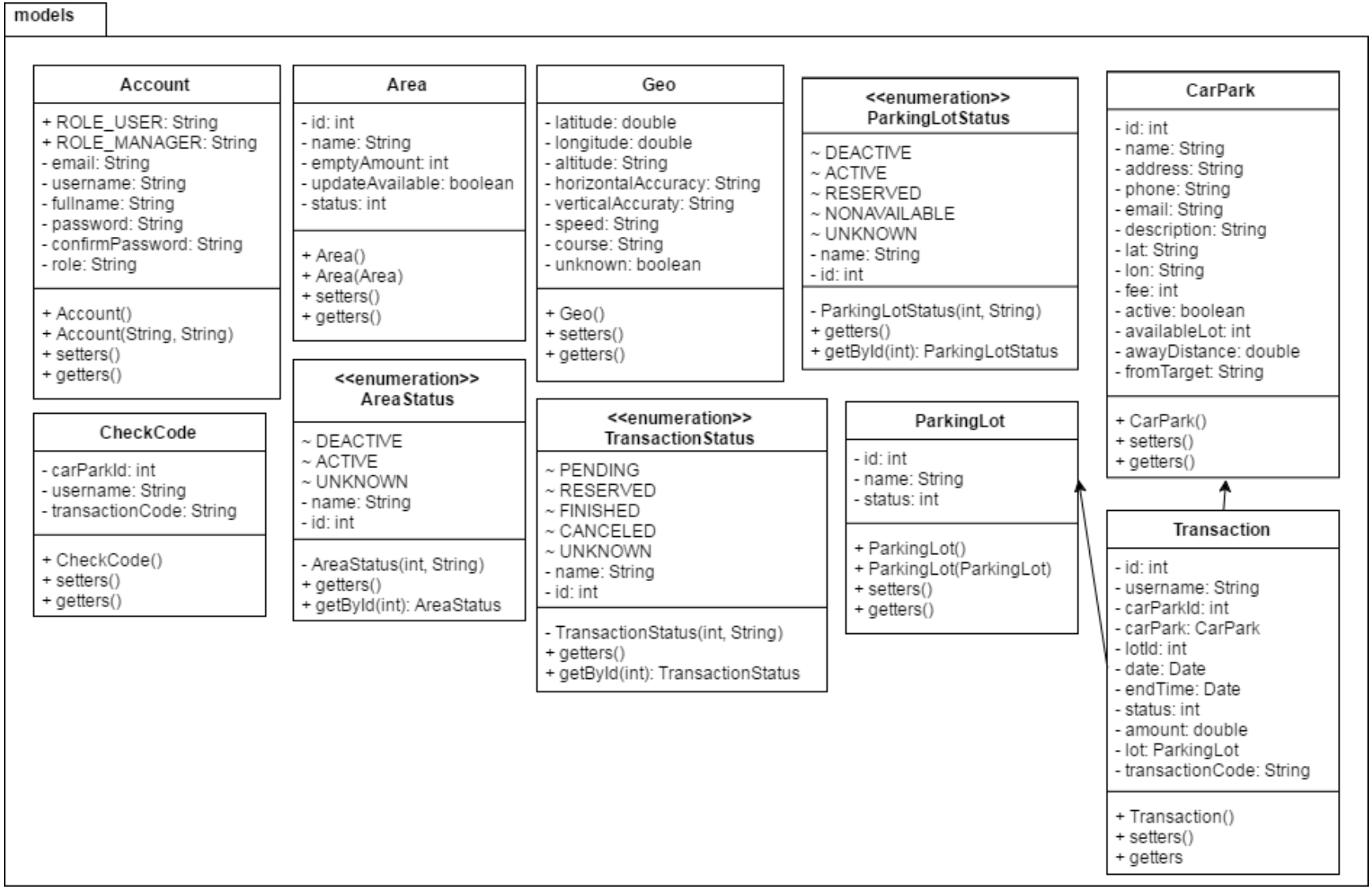


Figure 48: Models package class diagram

Class Dictionary: describes class	
Class Name	**Description**
Account	Contain the account information
Area	Contain the area information
CheckCode	Contain the check code information
Geo	Contain the geo information
CarPark	Contain the car park information
Transaction	Contain the transaction information
ParkingLot	Contain the parking lot information
TransactionStatus	An enumeration contains the status of transaction
AreaStatus	An enumeration contains the status of area
ParkingLotStatus	An enumeration contains the status of parking lot

Table 50: Models package class dictionary

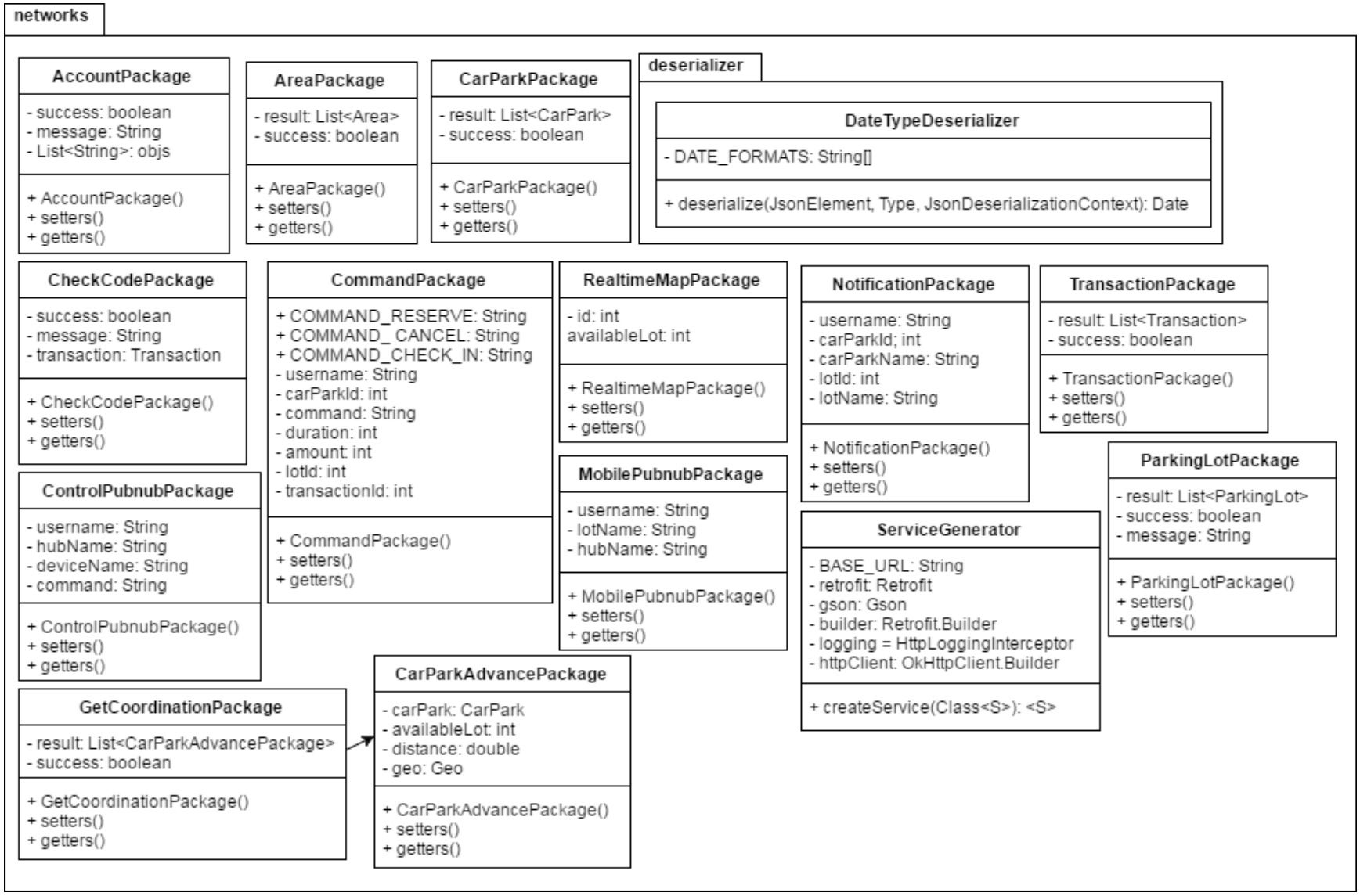


Figure 49: Networks package class diagram

Class Dictionary: describes class	
Class Name	**Description**
AccountPackage	Contain the account information used in network
AreaPackage	Contain the area information used in network
CarParkPackage	Contain the car park information used in network
CheckCodePackage	Contain the check code information used in network
CommandPackage	Contain the command information used in network
RealtimeMapPackage	Contain the realtime map information used in network
ControlPubnubPackage	Contain the control pubnub information used in network
MobilePubnubPackage	Contain the mobile pubnub information used in network
GetCoordinationPackage	Contain the get coordination information used in network
CarParkAdvancePackage	Contain the car park advance information used in network
NotificationPackage	Contain the notification information used in network
TransactionPackage	Contain the transaction information used in network
ParkingLotPackage	Contain the parking lot information used in network
ServiceGenerator	A generator to generate retrofit service with interfaces
DateTypeDeserializer	A deserializer that can deserialize multiple type of Date format in json

Table 51: Networks package class dictionary

4.2.2. Class Diagram Explanation

4.2.2.1. [Activities] MainActivity

Attributes

N/A

Methods

Method	Return type	Visibility	Description
onCreate	Void	Protected	Initialize activity
directActivity	Void	Private	Direct flow of activity depend on shared preferences status

Table 52: [Activities] MainActivity methods

4.2.2.2. [Activities] CarParkDetailActivity

Attributes

Attribute	Type	Visibility	Description
EXTRA_CAR_PARK	String	private	Extra name used in Intent
account	Account	private	Login user account
pubNub	PubNub	private	Instance of PubNub
toolbar	Toolbar	package	Instance of Toolbar
tvName	TextView	package	View of car park name
tvDistanceAway	TextView	package	View of distance between car park and target
tvAvailableLot	TextView	package	View of car park available lot
tvAddress	TextView	package	View of car park address
tvPhone	TextView	package	View of car park phone
tvEmail	TextView	package	View of car park email
tvDescription	TextView	package	View of car park description
btnCall	Button	package	View of call button
btnRoute	Button	package	View of request routing button
btnReserve	Button	package	View of request reservation button
carpark	CarPark	package	Object of CarPark

Table 53: [Activities] CarParkDetailActivity attributes

Methods

Method	Return type	Visibility	Description
createIntent	Intent	public	Return Intent of activity
onCreate	void	protected	Initialize activity
onOptionsItemSelected	boolean	public	Callback when options item is selected
initiateFields	void	private	Initialize all member variables
initiateViews	void	private	Initialize all views

setToolbarBackground	void	private	Set background for toolbar
onCallButtonClick	void	protected	Callback when call button is clicked
onRouteButtonClick	void	protected	Callback when route button is clicked
onReserveButtonClick	void	protected	Callback when reserve butotn is clicked
finalConfirm	void	private	Show final confirmation of request
reserveParkingLot	void	private	Process to reserve a parking lot
initatePubnub	void	private	Initalize PubNub instance
isBetween	boolean	private	Check if the number is between 2 other numbers
getDistanceString	String	private	Return distance in string format

Table 54: [Activities] CarParkDetailActivity methods

4.2.2.3. [Activities] ManagerActivity

Attributes

Attribute	Type	Visibility	Description
toolbar	Toolbar	protected	Instance of Toolbar
toolbarProgress	ProgressBar	protected	Instance of ProgressBar
drawerLayout	DrawerLayout	protected	Instance of DrawerLayout
navigationView	NavigationView	protected	Instance of NavigationView
rvCarParkList	RecyclerView	protected	Recycler View of car park list
detailDialog	MaterialDialog	private	Detail dialog of car park
etName	EditText	private	View of car park name in detail dialog
etPhone	EditText	private	View of car park phone in detail dialog
etEmail	EditText	private	View of car park email in detail dialog
etAddress	EditText	private	View of car park address in detail dialog
etDescription	EditText	private	View of car park description in detail dialog
focusedCarPark	CarPark	private	Instance of focused car park
btnPositive	MDButton	private	View of positive button of detail dialog
checkCodeDialog	MaterialDialog	private	Check code dialog
spnCarParks	Spinner	private	Drop down list of car parks in check code dialog

etUsername	EditText	private	View of username in check code dialog
etPin	EditText	private	View of pin code in check code dialog
drawerToggle	ActionBarDrawerToggle	private	Instance of ActionBarDrawerToggle
carParks	List<CarPark>	private	List of CarPark
adapter	CarParkAdapter	private	Adapter of CarPark

Table 55: [Activities] ManagerActivity attributes

Methods

Method	Return type	Visibility	Description
onCreate	void	protected	Initialize activity
onOptionsItemSelected	boolean	public	Callback when options item is selected
onPostCreate	void	protected	Callback after activity is created
onConfigurationChanged	void	public	Callback when activity change configuration
initiateFields	void	private	Initialize member variables
initiateViews	void	private	Initialize views
setupDetailDialog	void	private	Initialize detail dialog
setupCheckCodeDialog	void	private	Initialize check code dialog
onDrawerItemClick	void	private	Callback when drawer item is clicked
getCarParkData	void	private	Get car park data using API
updateCarPark	void	private	Update car park data using API
checkCode	void	private	Check username and pin code using API

Table 56: [Activities] ManagerActivity methods

4.2.2.4. [Activities] AreaListActivity

Attributes

Attribute	Type	Visibility	Description
EXTRA_CAR_PARK_ID	String	private	Extra name of intent
EXTRA_CAR_PARK_NAME	String	private	Extra name of intent
toolbar	Toolbar	protected	Instance of Toolbar
toolbarProgress	ProgressBar	protected	Instance of ProgressBar
rvArea	RecyclerView	protected	Recycler View of area list
detailDialog	MaterialDialog	private	Detail dialog of area
etName	EditText	private	View of area name in detail dialog
scActive	SwitchCompat	private	View of area active switch in detail dialog
focusedArea	Area	private	Instance of focused Area

btnPositive	MDButton	private	View of positive button in detail dialog
carParkId	int	private	Value of car park id
carParkName	String	private	Value of car park name
areas	List<Area>	private	List of Area
adapter	AreaAdapter	private	Adapter of Area

Table 57: [Activities] AreaListActivity attributes

Methods

Method	Return type	Visibility	Description
createIntent	Intent	public	Return intent of activity
onCreate	void	protected	Initialize activity
onOptionsItemSelected	boolean	public	Callback when options item is selected
initiateFields	void	private	Initialize member variables
initiateViews	void	private	Initialize views
getAreaData	void	private	Get area data using API
updateArea	void	private	Check change of area and call correct update method
updateAreaStatus	void	private	Update area status using API
updateAreaName	void	private	Update area name using API

Table 58: [Activities] AreaListActivity methods

4.2.2.5. [Activities] ParkingLotListActivity

Attributes

Attribute	Type	Visibility	Description
EXTRA_AREA_ID	String	private	Extra of intent
EXTRA_AREA_NAME	String	private	Extra of intent
toolbar	Toolbar	protected	Instance of Toolbar
toolbarProgress	ProgressBar	protected	Instance of ProgressBar
rvParkingLot	RecyclerView	protected	Recycler View of parking lot list
dialog	MaterialDialog	private	Detail dialog of parking lot
etName	EditText	private	View of parking lot name in detail dialog
tvStatus	TextView	private	View of lot status in detail dialog
focusedLot	ParkingLot	private	Instance of focused Parking Lot
btnPositive	MDButton	private	View of positive button in detail dialog
areaId	int	private	Value of area id
areaName	String	private	Value of area name
lots	List<ParkingLot>	private	List of Parking Lot

adapter	ParkingLotAdapter	private	Adapter of Parking Lot

Table 59: [Activities] ParkingLotListActivity attributes

Methods

Method	Return type	Visibility	Description
createIntent	Intent	public	Return intent of activity
onCreate	void	protected	Initialize activity
onOptionsItemSelected	boolean	public	Callback when options item is selected
initiateFields	void	private	Initialize member variables
initiateViews	void	private	Initialize views
getParkingLotData	void	private	Get parking lot data using API
updateLot	void	private	Update lot information using API
updateLotStatus	void	private	Update lot status using API

Table 60: [Activities] ParkingLotListActivity methods

4.2.2.6. [Activities] LoginActivity

Attributes

Attribute	Type	Visibility	Description
etUsername	EditText	package	View of username
etPassword	EditText	package	View of password
btnSignIn	EditText	package	View of sign in button
tvRegister	TextView	package	View of register text
tvSignInGuest	TextView	package	View of sign in as guest text
dialog	MaterialDialog	private	Progress dialog
account	Account	private	Instance of Account

Table 61: [Activities] LoginActivity attributes

Methods

Method	Return type	Visibility	Description
onCreate	void	public	Initialize activity
onLoginButtonClick	void	protected	Callback when login button is clicked
onRegisterTextClick	void	protected	Callback when register text is clicked
onSignInGuestTextClick	void	protected	Callback when sign in as guest is clicked

Table 62: [Activities] LoginActivity methods

4.2.2.7. [Activities] RegisterActivity

Attributes

Attribute	Type	Visibility	Description
etEmail	EditText	package	View of email
etPassword	EditText	package	View of password

etConfirmPassword	EditText	package	View of confirm password
etUsername	EditText	package	View of username
etFullname	EditText	package	View of fullname
btnRegister	Button	package	View of register button
tvLogin	TextView	package	View of login text

Table 63: [Activities] RegisterActivity attributes

Methods

Method	Return type	Visibility	Description
onCreate	void	protected	Initialize activity
validateEmail	boolean	private	Check email format
validatePassword	boolean	private	Check password format
onRegisterButtonClick	void	protected	Callback when register button is clicked
onLoginTextClick	void	protected	Callback when login text is clicked

Table 64: [Activities] RegisterActivity methods

4.2.2.8. [Activities] UserActivity

Attributes

Attribute	Type	Visibility	Description
MAP_ZOOM_TO	int	private	Value of map zoom
REQUEST_CHECK_SETTING	int	private	Value of request check setting
map	GoogleMap	private	Instance of GoogleMap
centerLocation	LatLng	private	Location of map center
currentLocation	LatLng	private	Location of current postion
placeLocation	LatLng	private	Location of searched place
fromYou	boolean	private	Flag to check if the distance show on marker is from current location or from place
placeName	String	private	Name of searched place
autocompleFragment	Place Autocomplete Fragment	private	Instance of PlaceAutocompleteFragment
refreshCarParkData	boolean	private	Flag to check if need to refresh car park data on map
googleApiClient	GoogleApi Client	private	Instance of GoogleApiClient
markerMap	HashMap< Integer, Marker>	private	Map of marker and car park id
pubNub	PubNub	private	Instance of PubNub

drawerLayout	DrawerLayout	private	Instance of DrawerLayout
toolbar	Toolbar	private	Instance of Toolbar
toobarProgress	ProgressBar	private	Instance of ProgressBar
navigationMenu	NavigationView	private	Instance of NavigationView
drawerToggle	ActionBar DrawerToggle	private	Instance of ActionBarDrawerToggle
searchRange	double	private	Value of search range
numberOfCar	int	private	Value of number of car to request data from API

Table 65: [Activities] UserActivity attributes

Methods

Method	Return type	Visibility	Description
onCreate	void	protected	Initialize activity
onPostCreate	void	protected	Callback after activity is created
onConfigurationChanged	void	public	Callback after activity change configuration
onResume	void	protected	Callback when activity resume
onStart	void	protected	Callback when activity start
onStop	void	protected	Callback when activity stop
onDestroy	void	protected	Callback when activity destroy
initiateInstance	void	private	Initialize member variables and views
setupDrawerContent	void	private	Setup drawer
onDrawerItemClick	void	private	Callback when drawer item is clicked
onOptionsItemSelected	boolean	public	Callback when options item is selected
initiatePubnub	void	private	Initialize PubNub
onMapReady	void	public	Callback when google map is ready
onConnected	void	public	Callback when google client is connected
getCurrentLocation	void	private	Get current location
getCarParkData	void	private	Get car park data using API
onConnectionSuspended	void	public	Callback when connection is suspended
onConnectionFailed	void	public	Callback when connection is failed
onActivityResult	vodi	public	Callback when intent return result
locationSettingRequest	void	private	Request turn on location for app

Table 66: [Activities] UserActivity methods

4.2.2.9. [Activities] CarParkListActivity

Attributes

Attribute	Type	Visibility	Description
EXTRA_CURRENT _LOCATION_LAT	String	private	Extra for intent
EXTRA_CURRENT _LOCATION_LON	String	private	Extra for intent
EXTRA_FROM_TARGET	String	private	Extra for intent
rvCarParkList	RecyclerView	package	Recycler View for car park list
toolbar	Toolbar	package	Instance of Toolbar
toolbarProgress	ProgressBar	package	Instance of ProgressBar
currentPosition	LatLng	private	Current location
adapter	CarParkAdvance Adapter	private	Advance adapter of Car Park
carParks	List<CarPark>	private	List of CarPark
target	String	private	Name of target
numberOfCar	int	private	Value of number of car to request search

Table 67: [Activities] CarParkListActivity attributes

Methods

Method	Return type	Visibility	Description
createIntent	Intent	public	Return intent of activity
onCreate	void	protected	Initialize activity
onOptionsItemSelected	boolean	public	Callback when options item is selected
initiateViews	void	private	Initialize views
initiateFields	void	private	Initialize member variables
getCarParkData	void	private	Get car park data using API

Table 68: [Activities] CarParkListActivity methods

4.2.2.10. [Activities] TransactionActivity

Attributes

Attribute	Type	Visibility	Description
transactions	List<Transaction>	private	List of Transaction
adapter	TransactionAdapter	private	Adapter of Transaction
rvTransactionList	RecyclerView	package	Recycler View of transaction list
toolbar	Toolbar	package	Instance of Toolbar
toolbarProgress	ProgressBar	package	Instance of ProgressBar

Table 69: [Activities] TransactionActivity attributes

Methods

Method	Return type	Visibility	Description
onCreate	void	protected	Initialize activity
onOptionsItemSelected	boolean	public	Callback when options item is selected
initiateViews	void	private	Initialize views
initiateFields	void	private	Initialize fields
cancelTransaction	void	private	Callback when cancel transaction button is clicked
checkInTransaction	void	private	Callback when check in transaction button is clicked
getTransactionData	void	private	Get transaction data using API

Table 70: [Activities] TransactionActivity methods

4.2.2.11. [Adapters] AreaAdapter

Attributes

Attribute	Type	Visibility	Description
listener	AreaAdapter. OnItemClickListener	private	Interface of item click listener
context	Context	private	Context of adapter
areas	List<Area>	private	List of Area

Table 71: [Adapters] AreaAdapter attributes

Methods

Method	Return type	Visibility	Description
onCreateViewHolder	AreaAdapter. ViewHolder	public	Callback when creating view holder
onBindViewHolder	AreaAdapter. ViewHolder	public	Callback when binding data to view holder
getItemCount	int	public	Return number of items in list
setOnItemClickListener	void	public	Set listener callback for item
getContext	Context	private	Return adapter context

Table 72: [Adapters] AreaAdapter methods

4.2.2.12. [Adapters] AreaAdapter.OnItemClickListener

Attributes

N/A

Methods

Method	Return type	Visibility	Description
onItemClick	void	package	Callback when item in list is clicked
onItemLongClick	void	package	Callback when item in list is being hold

Table 73: [Adapters] AreaAdapter.OnItemClickListener methods

4.2.2.13. [Adapters] AreaAdapter.ViewHolder

Attributes

Attribute	Type	Visibility	Description
tvName	TextView	package	View of area name
tvAvailableLot	TextView	package	View of area available lot

Table 74: [Adapters] AreaAdapter.ViewHolder attributes

Methods

Method	Return type	Visibility	Description
bind	void	package	Bind data to view holder
getAvailableColor	int	private	Return color for available view
isBetween	boolean	private	Check if number is between 2 other numbers

Table 75: [Adapters] AreaAdapter.ViewHolder methods

4.2.2.14. [Adapters] CarParkAdapter

Attributes

Attribute	Type	Visibility	Description
listener	CarParkAdapter. OnItemClickListener	private	Interface of item click listener
context	Context	private	Context of adapter
carParks	List<CarPark>	private	List of CarPark

Table 76: [Adapters] CarParkAdapter attributes

Methods

Method	Return type	Visibility	Description
onCreateViewHolder	CarParkAdapter. ViewHolder	public	Callback when creating view holder
onBindViewHolder	CarParkAdapter. ViewHolder	public	Callback when binding data to view holder
getItemCount	int	public	Return number of items in list
setOnItemClickListener	void	public	Set listener callback for item
getContext	Context	private	Return adapter context

Table 77: [Adapters] CarParkAdapter methods

4.2.2.15. [Adapters] CarParkAdapter.OnItemClickListener

Attributes

N/A

Methods

Method	Return type	Visibility	Description

onItemClick	void	package	Callback when item in list is clicked
onItemLongClick	void	package	Callback when item in list is being hold

Table 78: [Adapter] CarParkAdapter.OnItemClickListener methods

4.2.2.16. [Adapters] CarParkAdapter.ViewHolder

Attributes

Attribute	Type	Visibility	Description
tvName	TextView	package	View of car park name
tvAddress	TextView	package	View of car park address
tvAvailableLot	TextView	package	View of car park available lot

Table 79: [Adapters] CarParkAdapter.ViewHolder attributes

Methods

Method	Return type	Visibility	Description
bind	void	package	Bind data to view holder
getAvailableColor	int	private	Return color for available view
isBetween	boolean	private	Check if number is between 2 other numbers

Table 80: [Adapters] CarParkAdapter.ViewHolder methods

4.2.2.17. [Adapters] ParkingLotAdapter

Attributes

Attribute	Type	Visibility	Description
listener	ParkingLotAdapter. OnItemClickListener	private	Interface of item click listener
context	Context	private	Context of adapter
lots	List<ParkingLot>	private	List of ParkingLot

Table 81: [Adapters] ParkingLotAdapter attributes

Methods

Method	Return type	Visibility	Description
onCreateViewHolder	ParkingLotAdapter. ViewHolder	public	Callback when creating view holder
onBindViewHolder	ParkingLotAdapter. ViewHolder	public	Callback when binding data to view holder
getItemCount	int	public	Return number of items in list
setOnItemClickListener	void	public	Set listener callback for item
getContext	Context	private	Return adapter context

Table 82: [Adapters] ParkingLotAdapter methods

4.2.2.18. [Adapters] ParkingLotAdapter.OnItemClickListener

Attributes

N/A

Methods

Method	Return type	Visibility	Description
onItemClick	void	package	Callback when item in list is clicked
onItemLongClick	void	package	Callback when item in list is being hold

Table 83: [Adapters] ParkingLotAdapter.OnItemClickListener methods

4.2.2.19. [Adapters] ParkingLotAdapter.ViewHolder

Attributes

Attribute	Type	Visibility	Description
tvName	TextView	package	View of parking lot name

Table 84: [Adapters] ParkingLotAdapter.ViewHolder attributes

Methods

Method	Return type	Visibility	Description
bind	void	package	Bind data to view holder

Table 85: [Adapters] ParkingLotAdapter.ViewHolder methods

4.2.2.20. [Adapters] CarParkAdvanceAdapter

Attributes

Attribute	Type	Visibility	Description
listener	CarParkAdvanceAdapter. OnItemClickListener	private	Interface of item click listener
context	Context	private	Context of adapter
carParks	List<CarPark>	private	List of CarPark

Table 86: [Adapters] CarParkAdvanceAdapter attributes

Methods

Method	Return type	Visibility	Description
onCreateViewHolder	CarParkAdvanceAdapter. ViewHolder	public	Callback when creating view holder
onBindViewHolder	CarParkAdvanceAdapter. ViewHolder	public	Callback when binding data to view holder
getItemCount	int	public	Return number of items in list
setOnItemClickListener	void	public	Set listener callback for item
getContext	Context	private	Return adapter context

Table 87: [Adapters] CarParkAdvanceAdapter methods

4.2.2.21. [Adapters] CarParkAdvanceAdapter.OnItemClickListener

Attributes

N/A

Methods

Method	Return type	Visibility	Description
onItemClick	void	package	Callback when item in list is clicked
onItemLongClick	void	package	Callback when item in list is being hold

Table 88: [Adapters] CarParkAdvanceAdapter.OnItemClickListener methods

4.2.2.22. [Adapters] CarParkAdvanceAdapter.ViewHolder

Attributes

Attribute	Type	Visibility	Description
tvName	TextView	package	View of car park name
tvAvailableLot	TextView	package	View of car park available lot
tvDistance	TextView	package	View of car park distance
tvAddress	TextView	package	View of car park address

Table 89: [Adapters] CarParkAdvanceAdapter.ViewHolder attributes

Methods

Method	Return type	Visibility	Description
bind	void	package	Bind data to view holder
getAvailableColor	int	private	Return color for available view
isBetween	boolean	private	Check if number is between 2 other numbers
getDistanceString	String	private	Get distance in string format

Table 90: [Adapters] CarParkAdvanceAdapter.ViewHolder methods

4.2.2.23. [Adapters] TransactionAdapter

Attributes

Attribute	Type	Visibility	Description
listener	TransactionAdapter. OnItemClickListener	private	Interface of item click listener
context	Context	private	Context of adapter
transactions	List<Transaction>	private	List of Transaction

Table 91: [Adapters] TransactionAdapter attributes

Methods

Method	Return type	Visibility	Description

onCreateViewHolder	TransactionAdapter.ViewHolder	public	Callback when creating view holder
onBindViewHolder	TransactionAdapter.ViewHolder	public	Callback when binding data to view holder
getItemCount	int	public	Return number of items in list
setOnItemClickListener	void	public	Set listener callback for item
getContext	Context	private	Return adapter context

Table 92: [Adapters] TransactionAdapter methods

4.2.2.24. [Adapters] TransactionAdapter.OnItemClickListener

Attributes

N/A

Methods

Method	Return type	Visibility	Description
onItemClick	void	package	Callback when item in list is clicked
onItemLongClick	void	package	Callback when item in list is being hold

Table 93: [Adapters] TransactionAdapter.OnItemClickListener methods

4.2.2.25. [Adapters] TransactionAdapter.ViewHolder

Attributes

Attribute	Type	Visibility	Description
tvAddress	TextView	package	View of transaction address
tvDate	TextView	package	View of transaction date
tvStatus	TextView	package	View of transaction status
tvAmount	TextView	package	View of transaction amount
tvLotName	TextView	package	View of transaction lot name
tvPin	TextView	package	View of transaction pin code

Table 94: [Adapters] TransactionAdapter.ViewHolder attributes

Methods

Method	Return type	Visibility	Description
bind	void	package	Bind data to view holder
getLotNameText	String	private	Get lot name in string format
getAmountText	String	private	Get amount in string format
getFormatedDate	String	private	Get date in string format

Table 95: [Adapters] TransactionAdapter.ViewHolder methods

4.2.2.26. [Adapters] UserInfoWindowAdapter

Attributes

Attribute	Type	Visibility	Description

contentView	View	private	Content view
tvCarParkName	TextView	package	View of car park name
tvCarParkAddress	TextView	package	View of car park address
tvAwayDistance	TextView	package	View of car park away distance

Table 96: [Adapters] UserInfoWindowAdapter attributes

Methods

Method	Return type	Visibility	Description
getInfoWindow	View	public	Return Info Window view
getInfoContents	View	public	Return Info Contents view
getDistanceString	String	private	Get distance in string format

Table 97: [Adapters] UserInfoWindowAdapter methods

4.2.2.27. [Helpers] AppDatabaseHelper

Attributes

Attribute	Type	Visibility	Description
DATABASE_NAME	String	private	Name of database
DATABASE_VERSION	int	private	Version number of database
TABLE_TRANSACTION	String	private	Name of transaction table
KEY_TRANSACTION_ID	String	private	Key of transaction id
KEY_TRANSACTION _USERNAME	String	private	Key of transaction username
KEY_TRANSACTION _DATE	String	private	Key of transaction date
KEY_TRANSACTION _STATUS	String	private	Key of transaction status
KEY_TRANSACTION _CAR_PARK_ID	String	private	Key of transaction car park id
KEY_TRANSACTION _CAR_PARK_NAME	String	private	Key of transaction car park name
KEY_TRANSACTION _LOT_ID	String	private	Key of transaction lot id
KEY_TRANSACTION _LOT_NAME	String	private	Key of transaction lot name
KEY_TRANSACTION _AMOUNT	String	private	Key of transaction amount

Table 98: [Helpers] AppDatabaseHelper attributes

Methods

Method	Return type	Visibility	Description
getInstance	AppDatabaseHelper	public	Return singleton instance of helper
onConfigure	void	public	Callback when database is configured

onCreate	void	public	Callback when database is created
onUpgrade	void	public	Callback when database is upgraded
addTransaction	void	public	Add transaction to database
addOrUpdateTransaction	void	public	Add or Update transaction to database
getAllTransactions	List<Transaction>	public	Return all transactions of user in database

Table 99: [Helpers] AppDatabaseHelper methods

4.2.2.28. [Helpers] PubnubHelper

Attributes

Attribute	Type	Visibility	Description
CHANNEL_REALTIME_MAP	String	public	Name of realtime map channel
CHANNEL_USER	String	public	Name of user channel
CHANNEL_NOTIFICATION	String	public	Name of notification channel
SUBSCRIBE_KEY	String	private	Pubnub subscribe key
PUBLISH_KEY	String	private	Pubnub publish key
CHANNELS_LIST	List<String>	private	List of pubnub channels

Table 100: [Helpers] PubnubHelper attributes

Methods

Method	Return type	Visibility	Description
getPubNub	PubNub	public	Return a singleton instance of PubNub

Table 101: [Helpers] PubnubHelper methods

4.2.2.29. [Helpers] AccountHelper

Attributes

Attribute	Type	Visibility	Description
ACCOUNT_KEY	String	private	Key name of account in shared preferences

Table 102: [Helpers] AccountHelper attributes

Methods

Method	Return type	Visibility	Description
save	void	public	Save account information in shared preferences
get	Account	public	Get account information from shared preferences

clear	void	public	Delete all account informations in shared preferences

Table 103: [Helpers] AccountHelper methods

4.2.2.30. [Helpers] MapMarkerHelper

Attributes

N/A

Methods

Method	Return type	Visibility	Description
getParkingMarker	Bitmap	public	Return a custom bitmap marker
convertToPixels	int	private	Convert dp to pixels
isBetween	boolean	private	Check if the number is between 2 other numbers

Table 104: [Helpers] MapMarkerHelper methods

4.2.2.31. [Helpers] InternetHelper

Attributes

N/A

Methods

Method	Return type	Visibility	Description
isConnected	boolean	public	Check if internet is available

Table 105: [Helpers] InternetHelper methods

4.2.2.32. [Interfaces] AreaClient

Attributes

N/A

Methods

Method	Return type	Visibility	Description
getAreaByCarParkId	Call<AreaPackage>	package	Interface for get area by car park id API
updateName	Call<AreaPackage>	package	Interface for update area name API
updateStatus	Call<AreaPackage>	package	Interface for update area status API

Table 106: [Interfaces] AreaClient methods

4.2.2.33. [Interfaces] TransactionClient

Attributes

N/A

Methods

Method	Return type	Visibility	Description
getTransactionByUsername	Call<Transaction Package>	package	Interface for get transactions by username API
checkCode	Call<Transaction Package>	package	Interface for check code API

Table 107: [Interfaces] TransactionClient methods

4.2.2.34. [Interfaces] CarParkClient

Attributes

N/A

Methods

Method	Return type	Visibility	Description
getCoordinate NearesCarPark	Call<Get CoordinatePackage>	package	Interface for get nearest car park API
getCoordinate NearestCarPark ByRange	Call<Get CoordinatePackage>	package	Interface for get nearest car park in range API
getCarPark ByUsername	Call<CarPark Package>	package	Interface for get car park by username API
update	Call<CarPark Package>	package	Interface for update car park API

Table 108: [Interfaces] CarParkClient methods

4.2.2.35. [Interfaces] AccountClient

Attributes

N/A

Methods

Method	Return type	Visibility	Description
login	Call<Account Package>	package	Interface for login API
register	Call<Account Package>	package	Interface for register API

Table 109: [Interfaces] AccountClient methods

4.2.2.36. [Interfaces] ParkingLotClient

Attributes

N/A

Methods

Method	Return type	Visibility	Description
getParkingLot ByAreaId	Call<Parking LotPackage>	package	Interface for get parking lot by area id API
updateStatus	Call<Parking LotPackage>	package	Interface for update parking lot status API
updateName	Call<Parking LotPackage>	package	Interface for update parking lot name API

Table 110: [Interfaces] ParkingLotClient methods

4.2.2.37. [Models] Account

Attributes

Attribute	Type	Visibility	Description
ROLE_USER	String	public	Value of role user
ROLE_MANAGER	String	public	Value of role manager
email	String	private	Email of account
username	String	private	Username of account
fullname	String	private	Fullname of account
password	String	private	Password of account
confirmPassword	String	private	Confirm password
role	String	private	Role of account

Table 111: [Models] Account attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 112: [Models] Account methods

4.2.2.38. [Models] Area

Attributes

Attribute	Type	Visibility	Description
id	int	private	Unique id of area
name	String	private	Name of area
emptyAmount	int	private	Current available lot of area
updateAvailable	boolean	private	Flag to check if area is updatable
status	int	private	Current status of area

Table 113: [Models] Area attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 114: [Models] Area methods

4.2.2.39. [Models] CheckCode

Attributes

Attribute	Type	Visibility	Description
carParkId	int	private	Unique id of car park
username	String	private	Username of user want to check in with code
transactionCode	String	private	Pin code come with transaction

Table 115: [Models] CheckCode attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 116: [Models] CheckCode methods

4.2.2.40. [Models] Geo

Attributes

Attribute	Type	Visibility	Description
latitude	double	private	Geo location latitude
longitude	double	private	Geo location longitude
altitude	String	private	Geo location altitude
horizontalAccuracy	String	private	Geo location horizontal accuracy
verticalAccuracy	String	private	Geo location vertical accuracy
speed	String	private	Geo location speed
course	String	private	Geo location course
unknown	boolean	private	Flag to check Geo location

Table 117: [Models] Geo attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 118: [Models] Geo methods

4.2.2.41. [Models] CarPark

Attributes

Attribute	Type	Visibility	Description
id	int	private	Unique id of car park
name	String	private	Name of car park
address	String	private	Address of car park
phone	String	private	Phone of car park
email	String	private	Email of car park

description	String	private	Description of car park
lat	String	private	Latitude of car park
lon	String	private	Longitude of car park
fee	int	private	Fee per hour of car park
active	boolean	private	Flag to check if car park is active
availableLot	int	private	Current available parking lot in car park
awayDistance	double	private	Away distance of car park from target
fromTarget	String	private	Target to check away distance

Table 119: [Models] CarPark attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 120: [Models] CarPark methods

4.2.2.42. [Models] Transaction

Attributes

Attribute	Type	Visibility	Description
id	int	private	Unique id of transaction
username	String	private	Username of transaction
carParkId	int	private	Car park id of transaction
carpark	CarPark	private	Instance of CarPark object in transaction
lotId	int	private	Parking Lot id of transaction
lot	ParkingLot	private	Instance of ParkingLot object in transaction
date	Date	private	The moment user request reservation
endTime	Date	private	The end of reservation
status	int	private	Status of reservation
amount	double	private	Cost of the reservation
transactionCode	String	private	Pin code of transaction

Table 121: [Models] Transaction attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 122: [Models] Transaction methods

4.2.2.43. [Models] ParkingLot

Attributes

Attribute	Type	Visibility	Description
id	int	private	Unique if of parking lot
name	String	private	Name of parking lot
status	int	private	Status of parking lot

Table 123: [Models] ParkingLot attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 124: [Models] ParkingLot methods

4.2.2.44. [Models] TransactionStatus

Attributes

Attribute	Type	Visibility	Description
PENDING	enum	package	Enumeration of transaction pending status
RESERVED	enum	package	Enumeration of transaction reserved status
FINISHED	enum	package	Enumeration of transaction finished status
CANCELED	enum	package	Enumeration of transaction canceled status
UNKNOWN	enum	package	Enumeration of transaction unknown status
name	String	private	Name of enum
id	int	private	Id of enum

Table 125: [Models] TransactionStatus attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
getById	TransactionStatus	public	Get enum by id

Table 126: [Models] TransactionStatus methods

4.2.2.45. [Models] AreaStatus

Attributes

Attribute	Type	Visibility	Description
DEACTIVE	enum	package	Enumeration of area deactive status
ACTIVE	enum	package	Enumeration of area active status

UNKNOWN	enum	package	Enumeration of area unknown status
name	String	private	Name of enum
id	int	private	Id of enum

Table 127: [Models] AreaStatus attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
getById	AreaStatus	public	Get enum by id

Table 128: [Models] AreaStatus methods

4.2.2.46. [Models] ParkingLotStatus

Attributes

Attribute	Type	Visibility	Description
DEACTIVE	enum	package	Enumeration of parking lot deactive status
ACTIVE	enum	package	Enumeration of parking lot active status
RESERVED	enum	package	Enumeration of parking lot reserved status
NONAVAILABLE	enum	package	Enumeration of parking lot non-available status
name	String	package	Name of enum
id	int	package	Id of enum

Table 129: [Models] ParkingLotStatus attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
getById	ParkingLotStatus	public	Get enum by id

Table 130: [Models] ParkingLotStatus methods

4.2.2.47. [Network] AccountPackage

Attributes

Attribute	Type	Visibility	Description
success	boolean	private	Flag to check status of package
message	String	private	Message from server
objs	List<String>	private	List of account role

Table 131: [Network] AccountPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value

Setter	Void	public	Set value of attribute

Table 132: [Network] AccountPackage methods

4.2.2.48. [Network] AreaPackage

Attributes

Attribute	Type	Visibility	Description
result	List<Area>	private	List of area returned from server
success	boolean	private	Flag to check status of package

Table 133: [Network] AreaPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 134: [Network] AreaPackage methods

4.2.2.49. [Network] CarParkPackage

Attributes

Attribute	Type	Visibility	Description
result	List<CarPark>	private	List of car park returned from server
success	boolean	private	Flag to check the success of package

Table 135: [Network] CarParkPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 136: [Network] CarParkPackage methods

4.2.2.50. [Network] CheckCodePackage

Attributes

Attribute	Type	Visibility	Description
success	boolean	private	Flag to check the success of package
message	String	private	Message of package
transaction	Transaction	private	Transaction object returned from server

Table 137: [Network] CheckCodePackage attributes

Methods

Method	Return type	Visibility	Description

Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 138: [Network] CheckCodePackage methods

4.2.2.51. [Network] CommandPackage

Attributes

Attribute	Type	Visibility	Description
COMMAND_RESERVE	String	public	Value of reserve command
COMMAND_CANCEL	String	public	Value of cancel command
COMAND_CHECK_IN	String	public	Value of check in command
username	String	private	Username request command
carParkId	int	private	Id of target car park
command	String	private	Command of package
duration	int	private	Duration of reservation if command is reserve
amount	int	private	Total cost of reservation if command is reserve
lotId	int	private	Id of parking lot if command is cancel or check in
transactionId	int	private	Id of transaction if command is cancel or check in

Table 139: [Network] CommandPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 140: [Network] CommandPackage methods

4.2.2.52. [Network] RealtimeMapPackage

Attributes

Attribute	Type	Visibility	Description
id	int	private	Id of car park
availableLot	int	private	Current available lot of car park

Table 141: [Network] RealtimeMapPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 142: [Network] RealtimeMapPackage methods

4.2.2.53. [Network] ControlPubnubPackage

Attributes

Attribute	Type	Visibility	Description
username	String	private	Username request command
hubName	String	private	Name of target hub
deviceName	String	private	Name of target device
command	String	private	Command of package

Table 143: [Network] ControlPubnubPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 144: [Network] ControlPubnubPackage methods

4.2.2.54. [Network] MobilePubnubPackage

Attributes

Attribute	Type	Visibility	Description
username	String	private	Username of package
lotName	String	private	Lot name of package
hubName	String	private	Hub name of package

Table 145: [Network] MobilePubnubPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 146: [Network] MobilePubnubPackage methods

4.2.2.55. [Network] GetCoordinationPackage

Attributes

Attribute	Type	Visibility	Description
result	List<CarPark AdvancePackage>	private	List of car park advance package
success	boolean	private	Flag to check status of package

Table 147: [Network] GetCoordinationPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 148: [Network] GetCoordinationPackage methods

4.2.2.56. [Network] CarParkAdvancePackage

Attributes

Attribute	Type	Visibility	Description
carpark	CarPark	private	Instance of car park object
availableLot	int	private	Current available lot in car park
distance	double	private	Calculated distance away from car park
geo	Geo	private	Instanc of geo object

Table 149: [Network] CarParkAdvancePackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 150: [Network] CarParkAdvancePackage methods

4.2.2.57. [Network] NotificationPackage

Attributes

Attribute	Type	Visibility	Description
username	String	private	Username of notification
carParkId	int	private	Id of car park in notification
carParkName	String	private	Name of car park
lotId	int	private	Id of reserved lot in notification
lotName	String	private	Name of reserved lot

Table 151: [Network] NotificationPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 152: [Network] NotificationPackage methods

4.2.2.58. [Network] TransactionPackage

Attributes

Attribute	Type	Visibility	Description
result	List<Transaction>	private	List of transaction returned from server
success	boolean	private	Flag to check status of package

Table 153: [Network] TransactionPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 154: [Network] TransactionPackage methods

4.2.2.59. [Network] ParkingLotPackage

Attributes

Attribute	Type	Visibility	Description
result	List<ParkingLot>	private	List of parking lot returned from server
success	boolean	private	Flag to check status of package
message	String	private	Message of package

Table 155: [Network] ParkingLotPackage attributes

Methods

Method	Return type	Visibility	Description
Getter	Attribute type	public	Get attribute value
Setter	Void	public	Set value of attribute

Table 156: [Network] ParkingLotPackage methods

4.2.2.60. [Network] ServiceGenerator

Attributes

Attribute	Type	Visibility	Description
BASE_URL	String	private	Base url of API server
retrofit	Retrofit	private	Instance of Retrofit 2
gson	Gson	private	Instance of Gson
builder	Retrofi.Builder	private	Builder to create retrofit instance
logging	HttpLoggin Interceptor	private	Logging object
httpClient	OkHttpClient. Builder	private	Builder to create http client instance

Table 157: [Network] ServiceGenerator attributes

Methods

Method	Return type	Visibility	Description
createService	Service type	public	Generate new instance of service interface

Table 158: [Network] ServiceGenerator methods

4.2.2.61. [Network] DateTypeDeserializer

Attributes

Attribute	Type	Visibility	Description
DATE_FORMAT	String[]	private	List of common date time format

Table 159: [Network] DateTypeDeserializer attributes

Methods

Method	Return type	Visibility	Description

deserialize	Date	public	Return date object from json

Table 160: [Network] DateTypeDeserializer methods

4.2.3. Interaction Diagram

4.2.3.1. Load car parks on map

Summary: This diagram shows the process of app loads car parks data from API server and display them on map as marker, and update their available lots in real time.

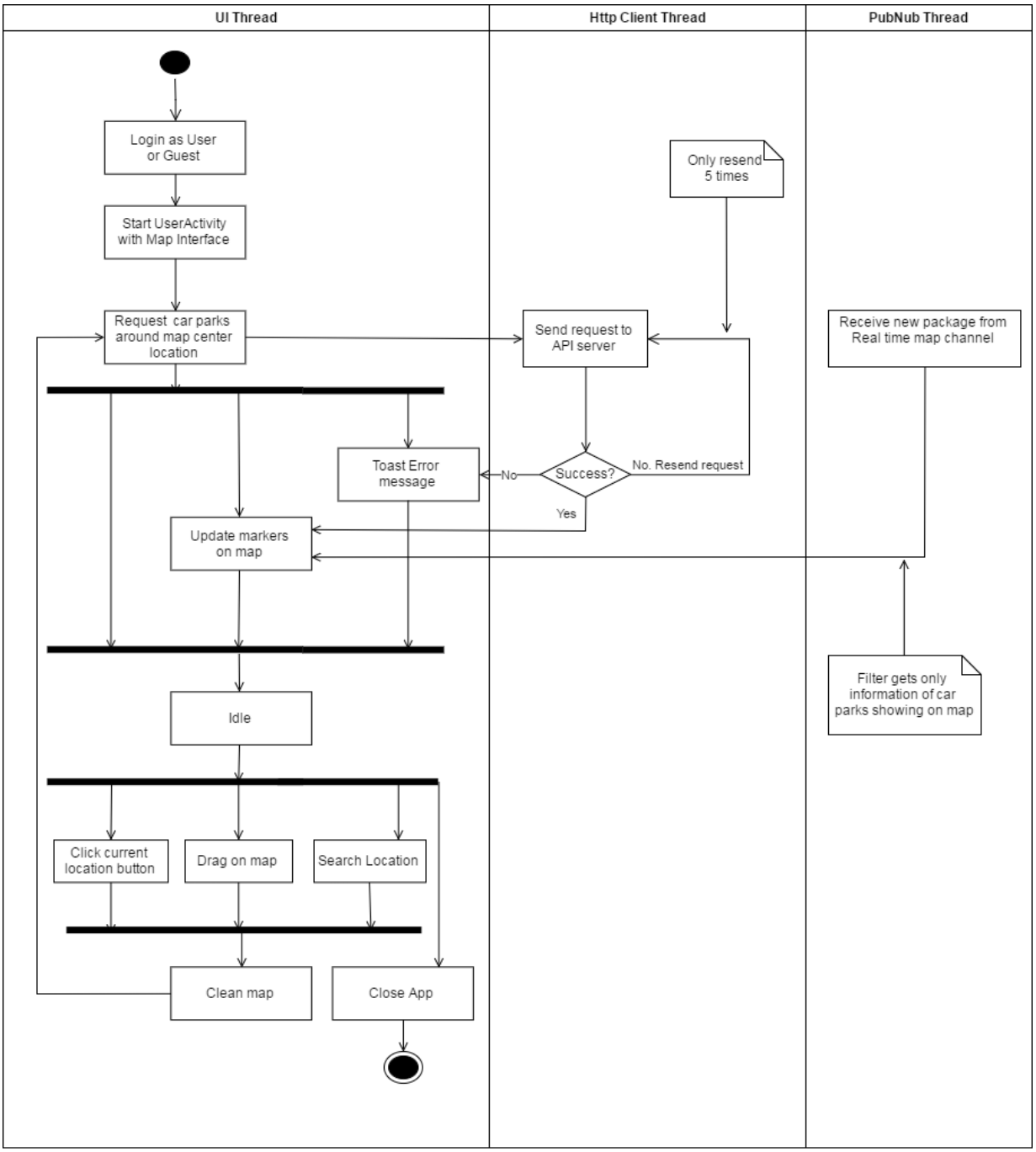


Figure 50: Interaction diagram - Load car parks on map

4.2.3.2. Load car parks in list

Summary: This diagram shows the process of app loads car parks data from API server and display them in list format, and update their available lots in real time.

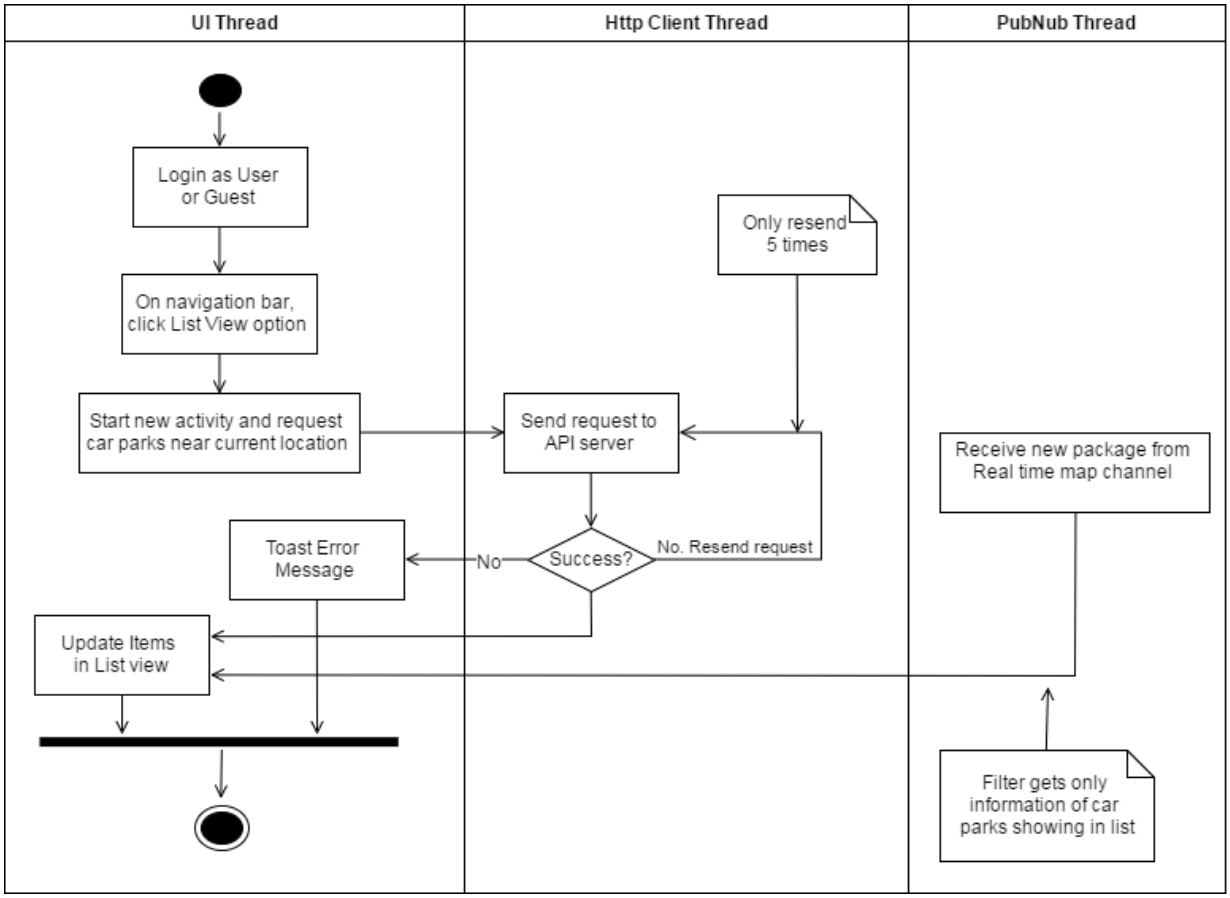


Figure 51: Interaction diagram - Load car parks in list

4.2.3.3. Car park detail information

Summary: This diagram shows the process of app loads car park detail information from API server and display it, update its available lots in real time and other action user can perform on the detail page.

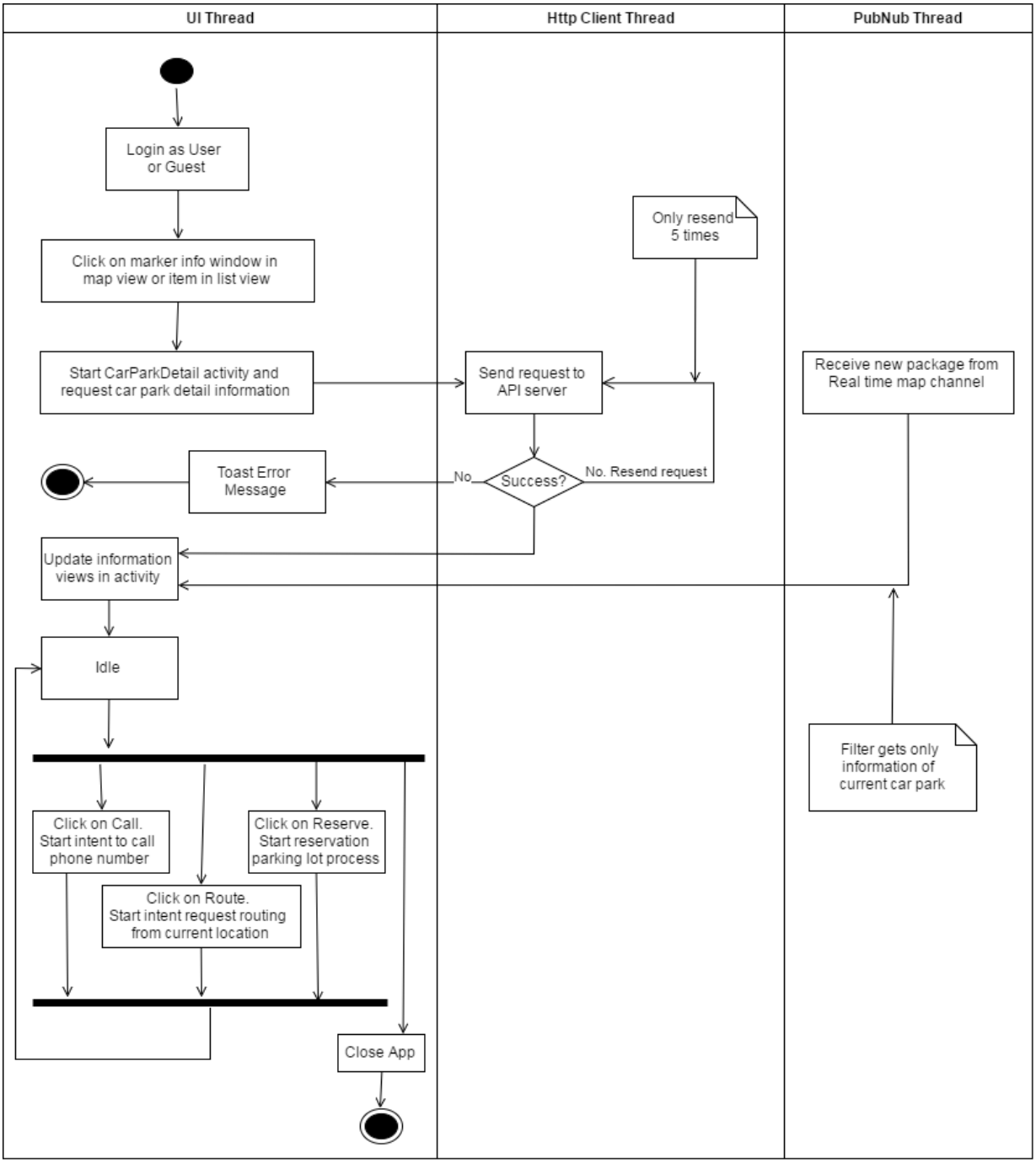


Figure 52: Interaction diagram - Car park detail information

4.2.3.4. Reserve parking lot process

Summary: This diagram shows the process of reserving parking lot, and receive notification about the reservation.

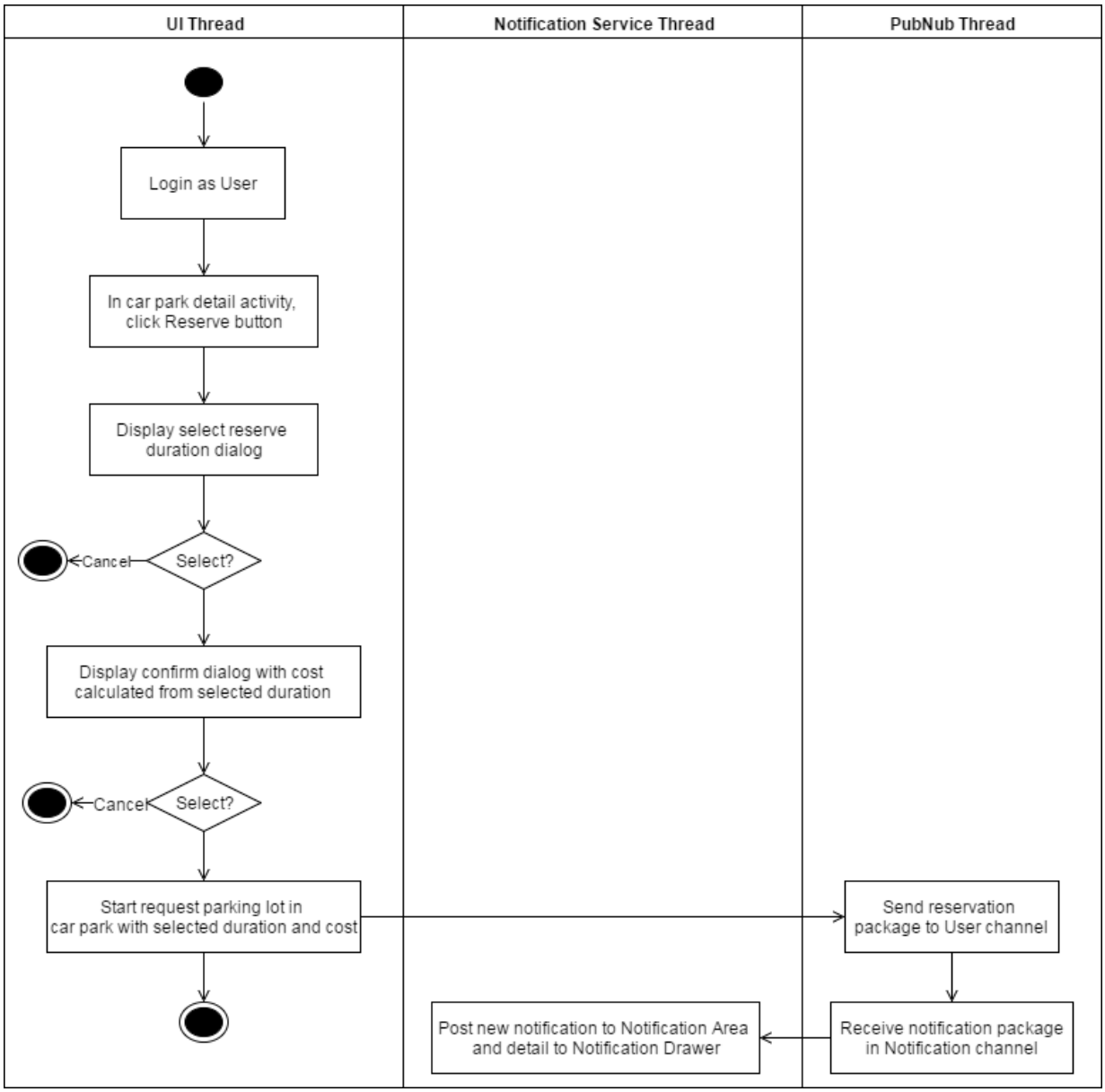


Figure 53: Interaction diagram - Reserve parking lot process

4.2.3.5. Transaction controller

Summary: This diagram shows the process of getting all transactions of user from API server and perform check-in, cancel process.

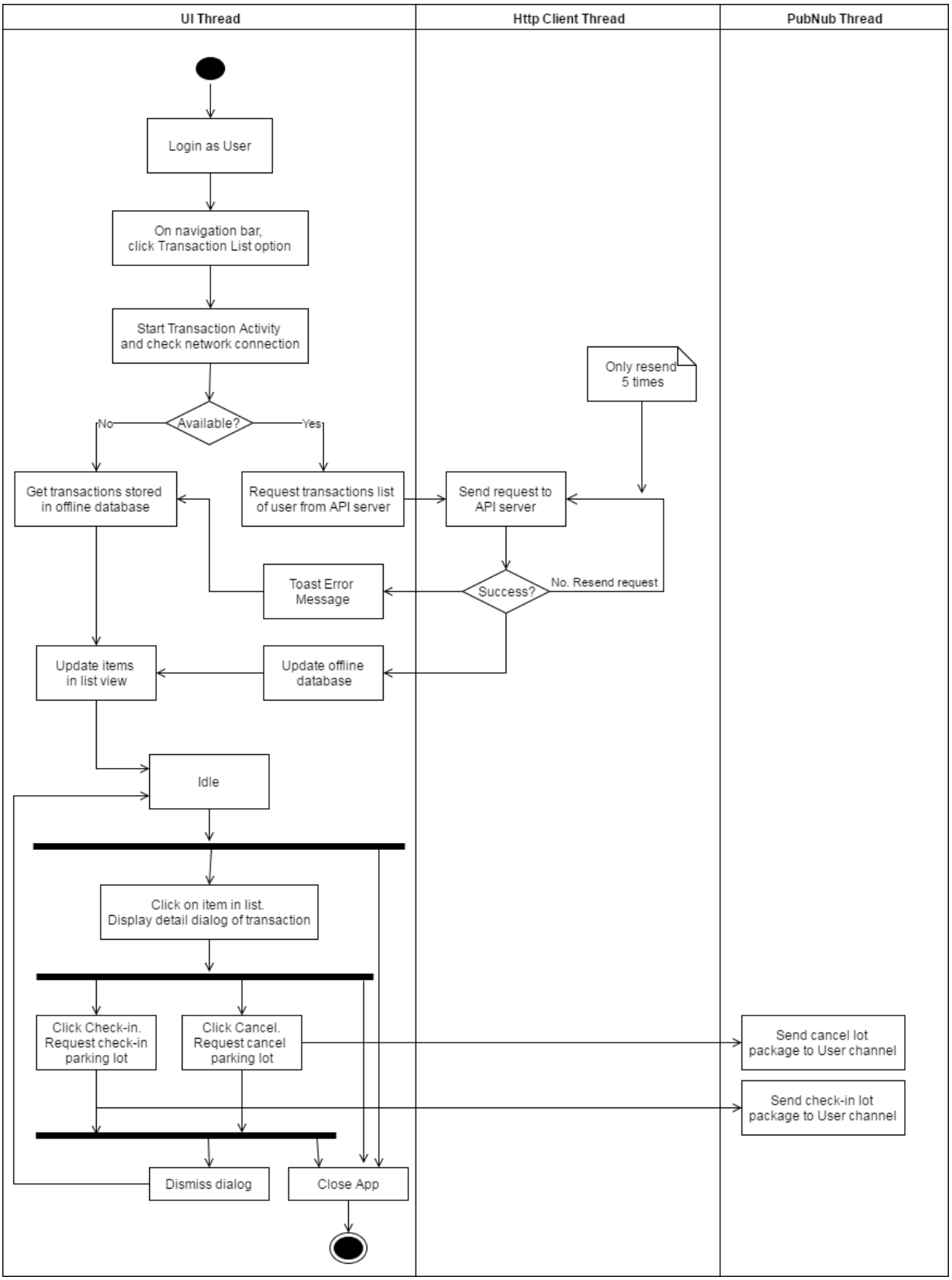


Figure 54: Interaction diagram - Transaction controller

4.2.3.6. Manage car parks

Summary: This diagram shows the process of app loads car parks from API server and display them in list from, update their available lots in real time, manage information and update to server and hub.

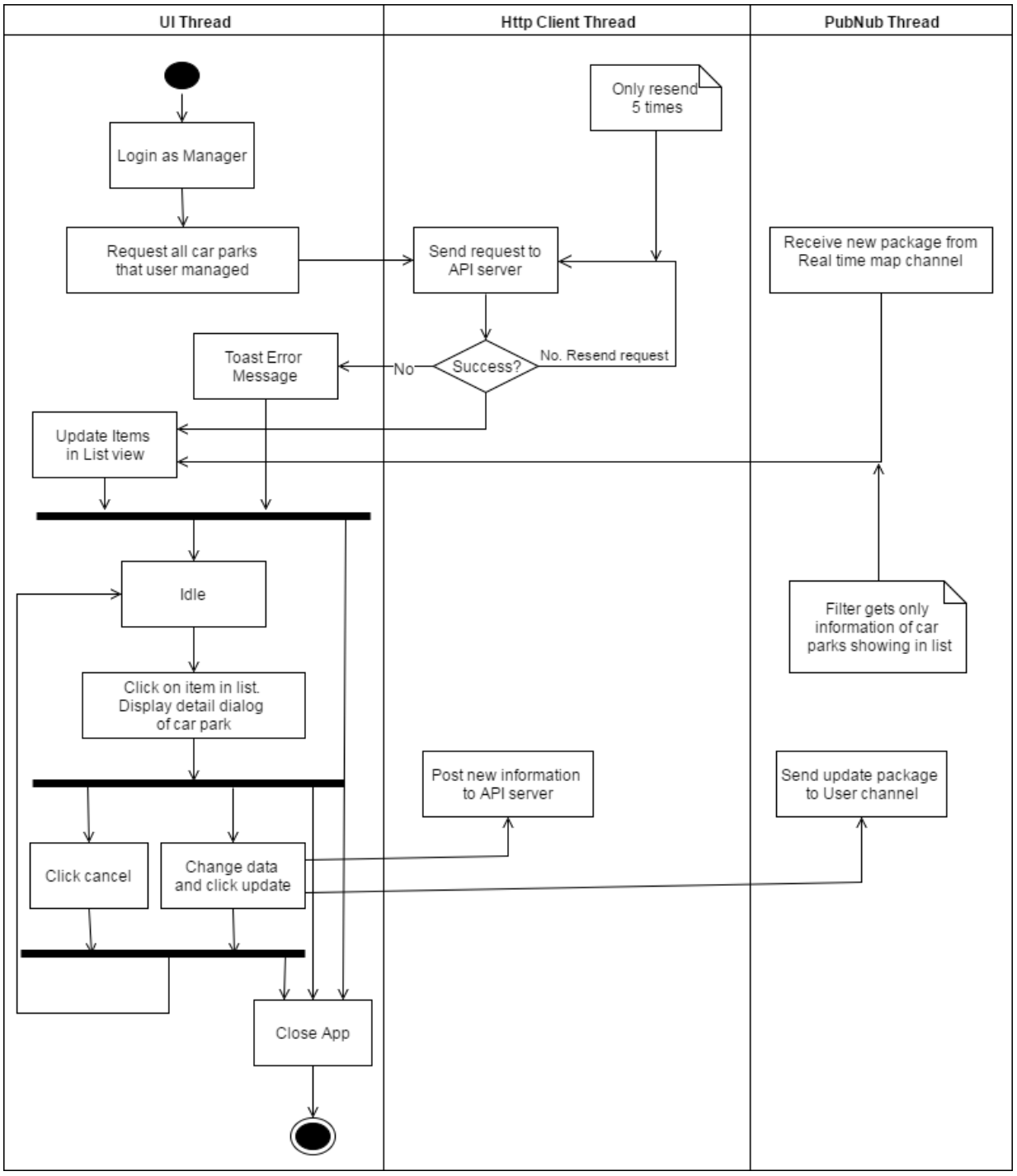


Figure 55: Interaction diagram - Manage car parks

4.2.3.7. Manage areas

Summary: This diagram shows the process of app loads areas from API server and display them in list from, manage information and update to server and hub.

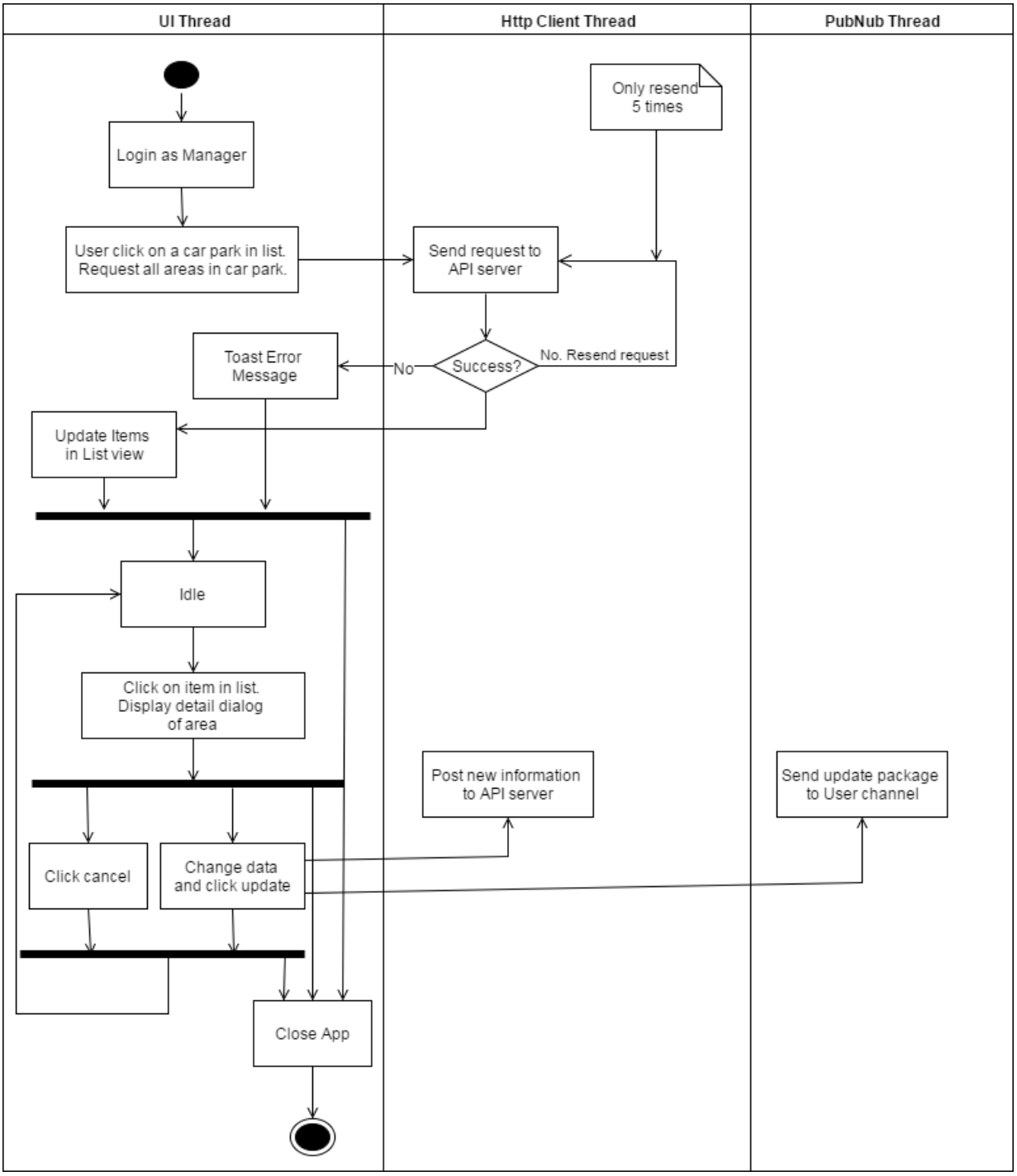


Figure 56: Interaction diagram - Manage areas

4.2.3.8. Manage parking lots

Summary: This diagram shows the process of app loads parking lot from API server and display them in list from, manage information and update to server and hub.

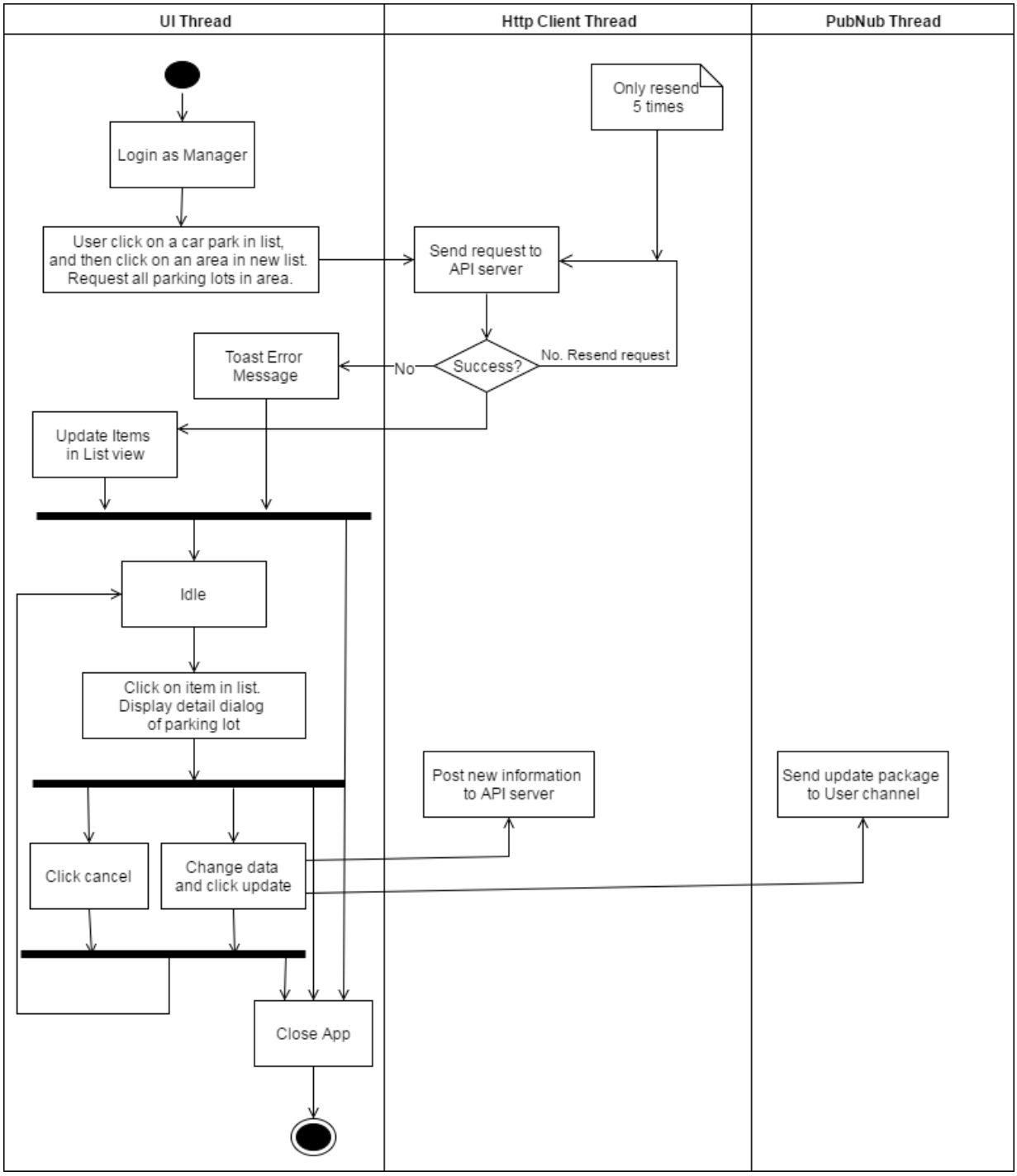


Figure 57: Interaction diagram - Manage parking lots

4.2.3.9. Check reservation PIN code

Summary: This diagram shows the process of how manager of car park can help user check-in their reservation.

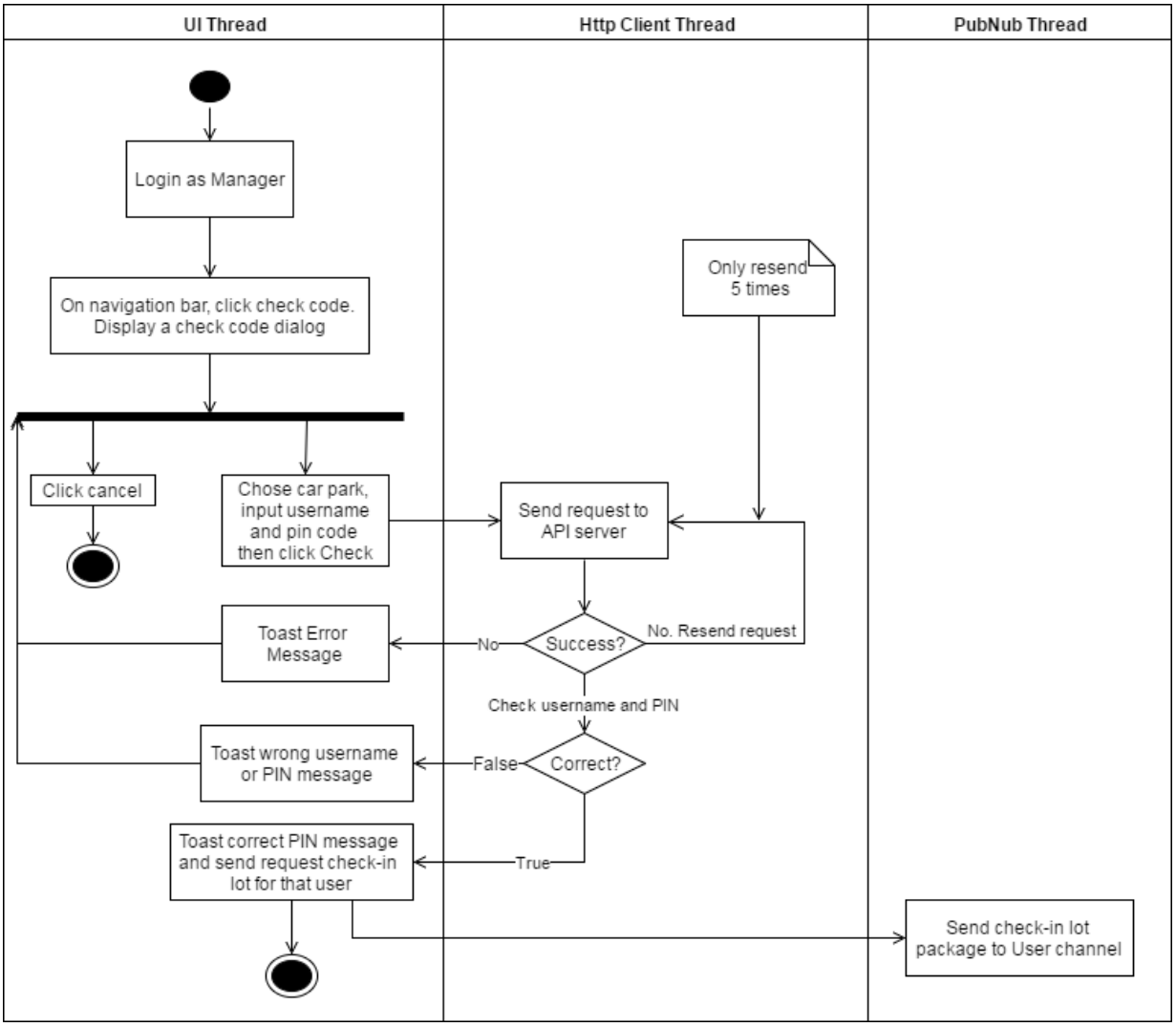


Figure 58: Interaction diagram - Check reservation PIN code

4.3. Embedded Program Detailed Description

4.3.1. Class Diagram

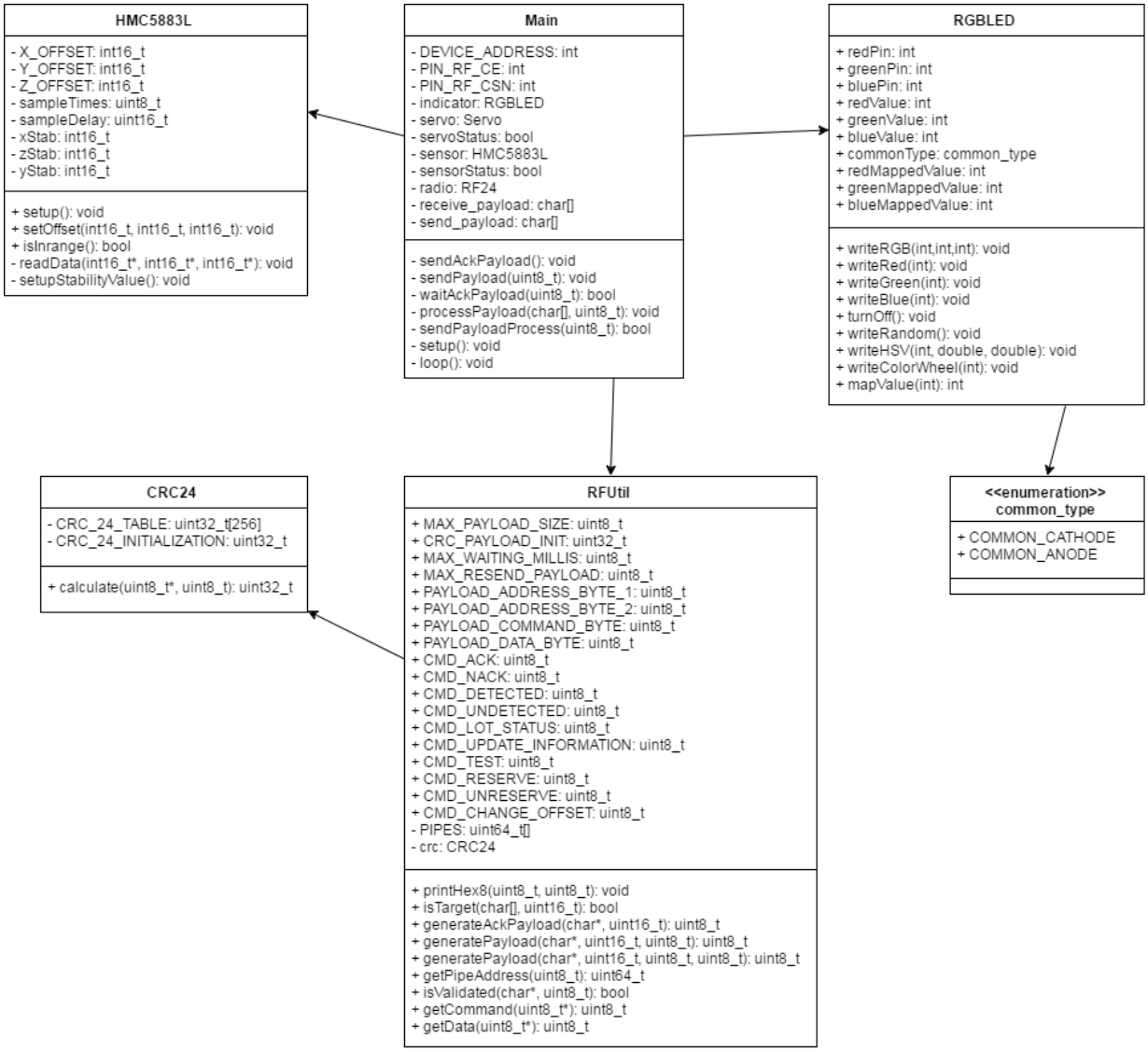


Figure 59: Lot node class diagram

Class Dictionary: describes class	
Class Name	**Description**
Main	Core program of node
RFUtil	Contain necessary information and methods support RF module
CRC24	Contain the checksum method for RFUtil
HMC5883L	Provides methods to use the magnetometer
RGBLED	Provides methods to control LED RGB
common_type	Enumeration contains type of LED RGB

Table 161: Lot node class dictionary

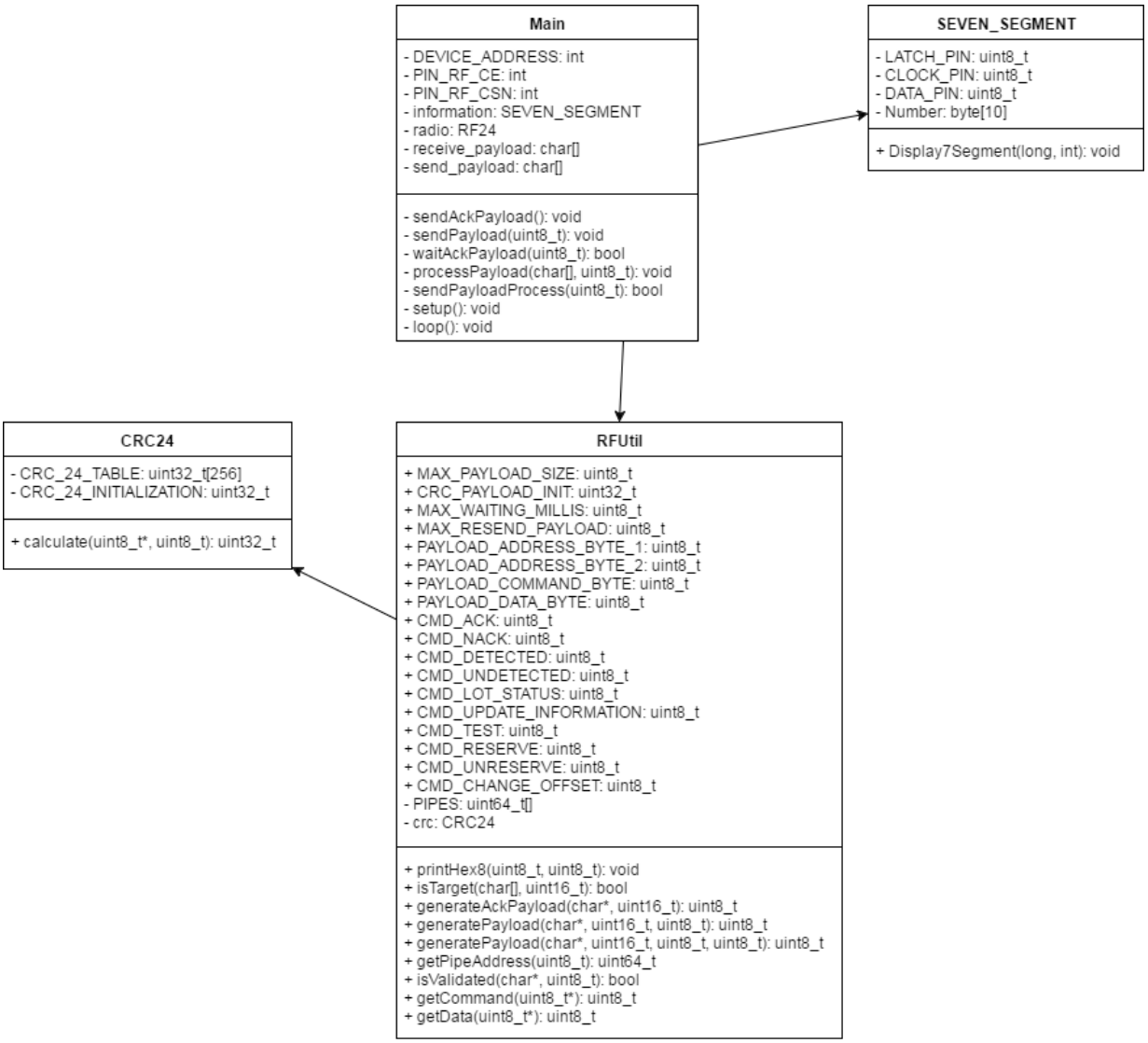


Figure 60: Sign node class diagram

Class Dictionary: describes class	
Class Name	**Description**
Main	Core program of node
RFUtil	Contain necessary information and methods support RF module
CRC24	Contain the checksum method for RFUtil
SEVEN_SEGMENT	Provide methods to use 7-segments sign

Table 162: Sign node class dictionary

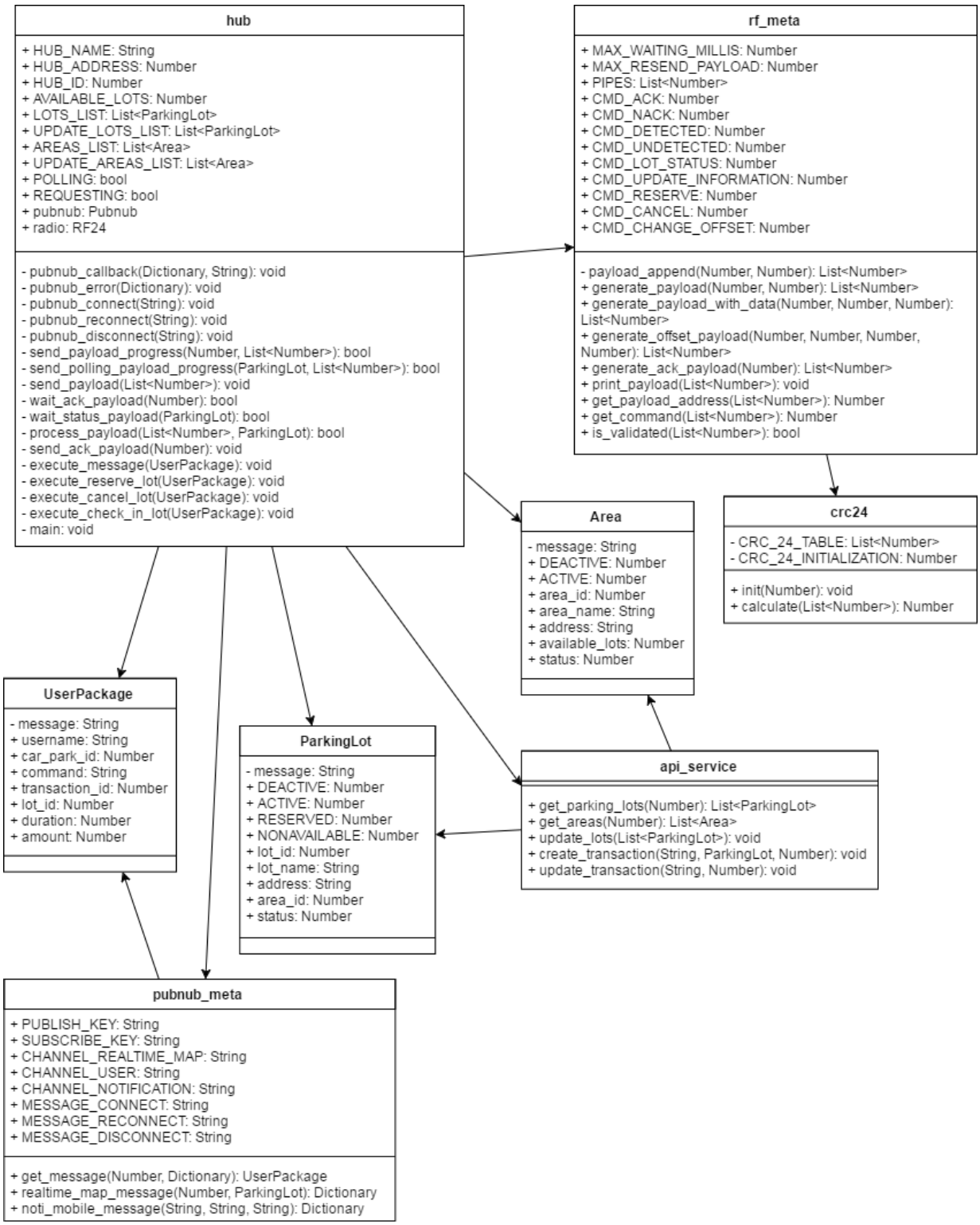


Figure 61: Hub class diagram

Class Dictionary: describes class	
Class Name	**Description**
hub	Main program of Hub
rf_meta	Contain information and necessary methods for RF
crc24	Contain the checksum method for rf_meta
ParkingLot	Information of parking lot
Area	Information of area
UserPackage	Information of user message in PubNub
api_service	Contain methods for API service
pubnub_meta	Contain methods for PubNub service

Table 163: Hub class dictionary

4.3.2. Class Diagram Explanation

4.3.2.1. [Lot] Main

Attributes

Attribute	Type	Visibility	Description
DEVICE_ADDRESS	int	private	Address of node
PIN_RF_CE	int	private	Pin used for CE pin of RF
PIN_RF_CSN	int	private	Pin used for CSN of RF
indicator	RGBLED	private	Instance of RGBLED
servo	Servo	private	Instance of Servo
servoStatus	bool	private	Flag to check if servo is running
sensor	HMC5883L	private	Instance of HMC5883L
sensorStatus	bool	private	Flag to check if the sensor detect car nearby
receive_payload	char[]	private	Received payload from RF
send_payload	char[]	private	Payload that will be send using RF

Table 164: [Lot] Main attributes

Methods

Method	Return type	Visibility	Description
sendAckPayload	void	private	Send ACK payload to Hub
sendPayload	void	private	Send payload to Hub
waiAckPayload	void	private	Wait for ACK payload from Hub
processPayload	void	private	Check received payload and perform work depend on the command
sendPayloadProcess	bool	private	Perform the process to send payload
setup	void	private	Run the setup whenever device start or reset
loop	void	private	Repeat until device stop

Table 165: [Lot] Main methods

4.3.2.2. [Lot] HMC5883L

Attributes

Attribute	Type	Visibility	Description
X_OFFSET	int16_t	private	Offset of X axis
Y_OFFSET	int16_t	private	Offset of Y axis
Z_OFFSET	int16_t	private	Offset of Z axis
sampleTimes	uint8_t	private	The number of times to sample to get stabilized axis
sampleDelay	uint16_t	private	The delay between sample
xStab	int16_t	private	Stabilized value of X axis
yStab	int16_t	private	Stabilized value of Y axis
zStab	int16_t	private	Stabilized value of Z axis

Table 166: [Lot] HMC5883L attributes

Methods

Method	Return type	Visibility	Description
setup	void	public	Run necessary methods to make the module work
setOffset	void	public	Set offset for the module to check if car is in range
isInRange	bool	public	Check if car is in range
readData	void	private	Get x,y,z axis from the module
setupStabilityValue	void	private	Perform sample multiple times to calculate stabilized axis

Table 167: [Lot] HMC5883L methods

4.3.2.3. [Lot] RGBLED

Attributes

Attribute	Type	Visibility	Description
redPin	int	public	LED pin
greenPin	int	public	LED pin
bluePin	int	public	LED pin
redValue	int	public	RGB value
greenValue	int	public	RGB value
blueValue	int	public	RGB value
commonType	common_type	public	LED type
redMappedValue	int	public	RGB value mapped based on commonType
greenMappedValue	int	public	RGB value mapped based on commonType
blueMappedValue	int	public	RGB value mapped based on commonType

Table 168: [Lot] RGBLED attributes

Methods

Method	Return type	Visibility	Description
writeRGB	void	public	Set all pins values
writeRed	void	public	Set just the red pin
writeGreen	void	public	Set just the green pin
writeBlue	void	public	Set just the blue pin
turnOff	void	public	Turn all pins off
writeRandom	void	public	Show a random color
writeHSV	void	public	Writes the LED based on HSV values
writeColorWheel	void	public	Show the HSV Color Wheel
mapValue	void	public	Map a value based on the common_type

Table 169: [Lot] RGBLED methods

4.3.2.4. [Lot] common_type

Attributes

Attribute	Type	Visibility	Description
COMMON_CATHODE	enum	public	Value of common cathode LED type
COMMON_ANODE	enum	public	Value of common anode LED type

Table 170: [Lot] common_type attributes

Methods

N/A

4.3.2.5. [Sign] Main

Attributes

Attribute	Type	Visibility	Description
DEVICE_ADDRESS	int	private	Address of node
PIN_RF_CE	int	private	Pin used for CE pin of RF
PIN_RF_CSN	int	private	Pin used for CSN of RF
information	SERVEN_SEGMENT	private	Instance of SEVEN_SEGMENT
receive_payload	char[]	private	Received payload from RF
send_payload	char[]	private	Payload that will be send using RF

Table 171: [Sign] Main attributes

Methods

Method	Return type	Visibility	Description
sendAckPayload	void	private	Send ACK payload to Hub

sendPayload	void	private	Send payload to Hub
waiAckPayload	void	private	Wait for ACK payload from Hub
processPayload	void	private	Check received payload and perform work depend on the command
sendPayloadProcess	bool	private	Perform the process to send payload
setup	void	private	Run the setup whenever device start or reset
loop	void	private	Repeat until device stop

Table 172: [Sign] Main methods

4.3.2.6. [Sign] SEVEN_SEGMENT

Attributes

Attribute	Type	Visibility	Description
LATCH_PIN	uint8_t	private	Latch pin of 595
CLOCK_PIN	uint8_t	private	Clock pin of 595
DATA_PIN	uint8_t	private	Data pin of 595
Number	byte[10]	private	Binary array for display 7-segments led using 595

Table 173: [Sign] SEVEN_SEGMENT attributes

Methods

Method	Return type	Visibility	Description
Display7Segment	void	public	Display available lot number to 7-segmens sign

Table 174: [Sign] SEVEN_SEGMENT methods

4.3.2.7. [Lot/Sign] RFUtil

Attributes

Attribute	Type	Visibility	Description
MAX_PAYLOAD_SIZE	uint8_t	public	Max size of payload
CRC_PAYLOAD_INIT	uint32_t	public	Initialize value to use with CRC24
MAX_WAITING_MILLIS	uint8_t	public	Max waiting ACK time in millis
MAX_RESEND_PAYLOAD	uint8_t	public	Max resend times
PAYLOAD_ADDRESS_ BYTE_1	uint8_t	public	Position of 1st address byte
PAYLOAD_ADDRESS_ BYTE_2	uint8_t	public	Position of 2nd address byte
PAYLOAD_COMMAND_ BYTE	uint8_t	public	Position of command byte
PAYLOAD_DATA_BYTE	uint8_t	public	Position of data byte

CMD_ACK	uint8_t	public	ACK command byte
CMD_NACK	uint8_t	public	NACK command byte
CMD_DETECTED	uint8_t	public	Detected command byte
CMD_UNDETECTED	uint8_t	public	Undetected command byte
CMD_LOT_STATUS	uint8_t	public	Lot status command byte
CMD_UPDATE _INFORMATION	uint8_t	public	Update information command byte
CMD_TEST	uint8_t	public	Test command byte
CMD_RESERVE	uint8_t	public	Reserve command byte
CMD_UNRESERVE	uint8_t	public	Unreserve command byte
CMD_CHANGE_OFFSET	uint8_t	public	Change offset command byte
PIPES	uint64_t[]	private	List of pipes address
crc	CRC24	private	Instance of CRC24

Table 175: [Lot/Sign] RFUtil attributes

Methods

Method	Return type	Visibility	Description
printHex8	void	public	Print payload as hex format
isTarget	bool	public	Check if the target of payload is current device
generateAckPayload	uint8_t	public	Create ACK payload. Return payload size
generatePayload	uint8_t	public	Create payload. Return payload size
getPipeAddress	uint64_t	public	Return pipe address from position
isValidated	bool	public	Check if payload satisfy CRC
getCommand	uint8_t	public	Return command byte from payload
getData	uint8_t	public	Return data byte from payload

Table 176: [Lot/Sign] RFUtil methods

4.3.2.8. [Lot/Sign] CRC24

Attributes

Attribute	Type	Visibility	Description
CRC_24_TABLE	uint32_t[256]	private	Table of CRC24 defined value
CRC_24 _INITIALIZATION	uint32_t	private	Initialization value for crc calculate

Table 177:: [Lot/Sign] CRC24 attributes

Methods

Method	Return type	Visibility	Description
calculate	uint32_t	public	Return the calculated checksum

Table 178: [Lot/Sign] CRC24 methods

4.3.2.9. [Hub] hub

Attributes

Attribute	Type	Visibility	Description
HUB_NAME	String	public	Name of car park
HUB_ADDRESS	Number	public	2 bytes address of hub
HUB_ID	Number	public	Id of car park
AVAILABLE_LOTS	Number	public	Current available lots in car park
LOTS_LIST	List<ParkingLot>	public	List of parking lots
UPDATE_ LOTS_LIST	List<ParkingLot>	public	List of lots need to update
AREAS_LIST	List<Area>	public	List of areas
UPDATE_ AREAS_LISST	List<Area>	public	List of areas need to update
POLLING	bool	public	Flag to check if the main progress is polling
REQUESTING	bool	public	Flag to check if there is a need to change main progress to execute message
pubnub	Pubnub	public	Instance of Pubnub
radio	RF24	public	Insantce of RF24

Table 179: [Hub] hub attributes

Methods

Method	Return type	Visibility	Description
pubnub_callback	void	private	Callback when a pubnub message is received
pubnub_error	void	private	Callback when there is a error in sending pubnub message
pubnub_connect	void	private	Callback when successfully connect to pubnub channel
pubnub_reconnect	void	private	Callback when need to reconnect channel
pubnub_disconnect	void	private	Callback when the connection with pubnub channel is cut
send_payload_progress	bool	private	Progress to send payload
send_polling_ payload_progress	bool	private	Progress to send polling payload
send_payload	void	private	Send payload using RF module
wait_ack_payload	bool	private	Wait for ack payload from target
wait_status_payload	bool	private	Wait for update status from lot
process_payload	bool	private	Check and process received payload
send_ack_payload	void	private	Send ACK payload to target

execute_message	void	private	Check and execute received pubnub message
execute_reserve_lot	void	private	Process to perform reserve lot action
execute_cancel_lot	void	private	Process to perform cancel lot action
execute_check_in_lot	void	private	Process to perform check in lot action
main	void	private	Perform program of main thread

Table 180: [Hub] hub methods

4.3.2.10. [Hub] rf_meta

Attributes

Attribute	Type	Visibility	Description
MAX_WAITING _MILLIS	Number	public	Max wait times for ACK
MAX_RESEND_ PAYLOAD	Number	public	Max times to resend payload
PIPES	List<Number>	public	List of pipes address for RF
CMD_ACK	Number	public	Byte of ACK command
CMD_NACK	Number	public	Byte of NACK command
CMD_DETECTED	Number	public	Byte of detected command
CMD_UNDETECTED	Number	public	Byte of undetected command
CMD_LOT_STATUS	Number	public	Byte of lot status command
CMD_UPDATE_ INFORMATION	Number	public	Byte of update information command
CMD_RESERVE	Number	public	Byte of reserve command
CMD_CANCEL	Number	public	Byte of cancel command
CMD_CHANGE_ OFFSET	Number	public	Byte of change offset command

Table 181: [Hub] rf_meta attributes

Methods

Method	Return type	Visibility	Description
payload_append	List<Number>	private	Add byte to payload
generate_payload	List<Number>	public	Create payload with target address and command
generate_payload_ with_data	List<Number>	public	Create payload with target address, command and data byte
generate_offset_ payload	List<Number>	public	Create special payload to update offset
generate_ack_ payload	List<Number>	public	Create ACK payload
print_payload	void	public	Print payload in hex form
get_payload_address	Number	public	Get address from payload

get_command	Number	public	Get command from payload
is_validated	bool	public	Check checksum of payload

Table 182: [Hub] rf_meta methods

4.3.2.11. [Hub] crc24

Attributes

Attribute	Type	Visibility	Description
CRC_24_TABLE	List<Number>	private	Table of CRC24 defined value
CRC_24 _INITIALIZATION	Number	private	Initialization value for crc calculate

Table 183: [Hub] crc24 attributes

Methods

Method	Return type	Visibility	Description
init	void	public	Set crc24 initialization to another value
calculate	Number	public	Return the calculated checksum

Table 184: [Hub] crc24 methods

4.3.2.12. [Hub] api_service

Attributes

N/A

Methods

Method	Return type	Visibility	Description
get_parking_lots	List<ParkingLot>	public	Get all active lots of car park from API server
get_areas	List<Area>	public	Get all active areas of car park from API server
update_lots	void	public	Update lot information using API
create_transaction	void	public	Create new transaction using API
update_transaction	void	public	Update transaction status using API

Table 185: [Hub] api_service methods

4.3.2.13. [Hub] pubnub_meta

Attributes

Attribute	Type	Visibility	Description
PUBLISH_KEY	String	public	Publish key for pubnub
SUBSCRIBE_KEY	String	public	Subscribe key for pubnub
CHANNEL _REALTIME_MAP	String	public	Key name of realtime map channel
CHANNEL_USER	String	public	Key name of user channel

CHANNEL _NOTIFICATION	String	public	Key name of notification channel
MESSAGE _CONNECT	String	public	Value of connect message
MESSAGE _RECONNECT	String	public	Value of reconnect message
MESSAGE _DISCONNECT	String	public	Value of disconnect message

Table 186: [Hub] pubnub_meta attributes

Methods

Method	Return type	Visibility	Description
get_message	UserPackage	public	Return user package from json
realtime_map _message	Dictionary	public	Create a json for realtime map channel
noti_mobile _message	Dictionary	public	Create a json for notification channel

Table 187: [Hub] pubnub_meta methods

4.3.2.14. [Hub] Area

Attributes

Attribute	Type	Visibility	Description
message	String	private	Format to string of object
DEACTIVE	Number	public	Value of deactive status
ACTIVE	Number	public	Value of active status
area_id	Number	public	Unique Id of area
area_name	String	public	Name of area
address	Number	public	Byte address of sign node
availables_lots	Number	public	Current available lots in area
status	Number	public	Status of area

Table 188: [Hub] Area attributes

Methods

N/A

4.3.2.15. [Hub] ParkingLot

Attributes

Attribute	Type	Visibility	Description
message	String	private	Format to string of object
DEACTIVE	Number	public	Value of deactive status
ACTIVE	Number	public	Value of active status
RESERVED	Number	public	Value of reserved status
NONAVAILABLE	Number	public	Value of non-available status
lot_id	Number	public	Unique id of parking lot

lot_name	String	public	Name of parking lot
address	Number	public	Byte address of lot node
area_id	Number	public	Unique Id of area contains lot
status	Number	public	Status of lot

Table 189: [Hub] ParkingLot attributes

Methods

N/A

4.3.2.16. [Hub] UserPackage

Attributes

Attribute	Type	Visibility	Description
message	String	private	Format to string of object
username	String	public	Username send the request
car_park_id	Number	public	Id of car park
command	String	public	Command of package
transaction_id	Number	public	Transaction Id if command is cancel or check in transaction
lot_id	Number	public	Lot node id if command is cancel or check in
duration	Number	public	Duration of the reservation if command is reserve
amount	Number	public	Cost of the reservation if command is reserve

Table 190: [Hub] UserPackage attributes

Methods

N/A

4.3.3. Interaction Diagram

4.3.3.1. Lot node process

Summary: This diagram shows the process of the MCU in lot node detects car, transmits and receives data from RF network and controls other actuator.

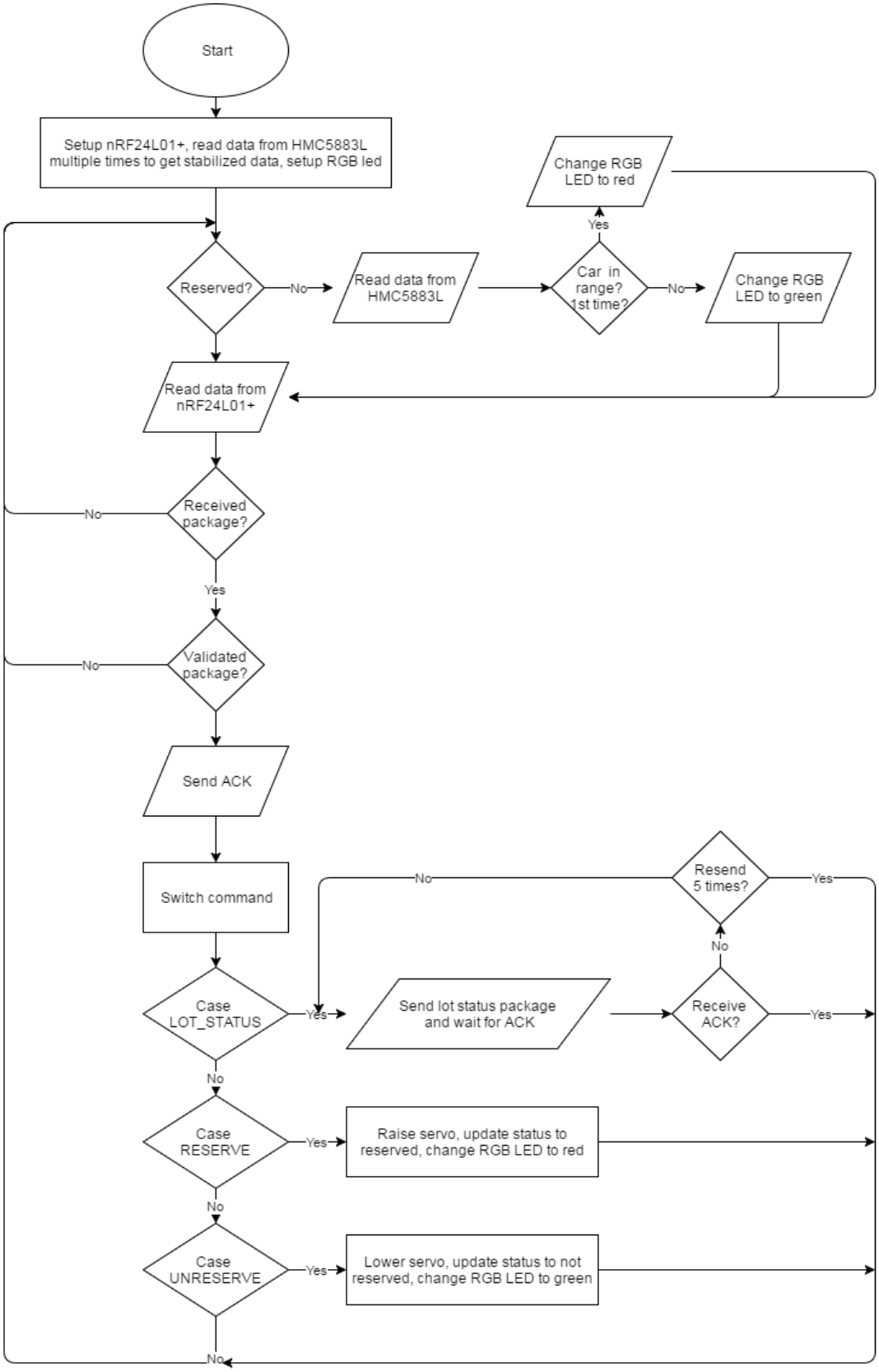


Figure 62: Interaction diagram - Lot node process

4.3.3.2. Sign node process

Summary: This diagram shows the process of the MCU in sign node transmits and receives data from RF network and controls led board.

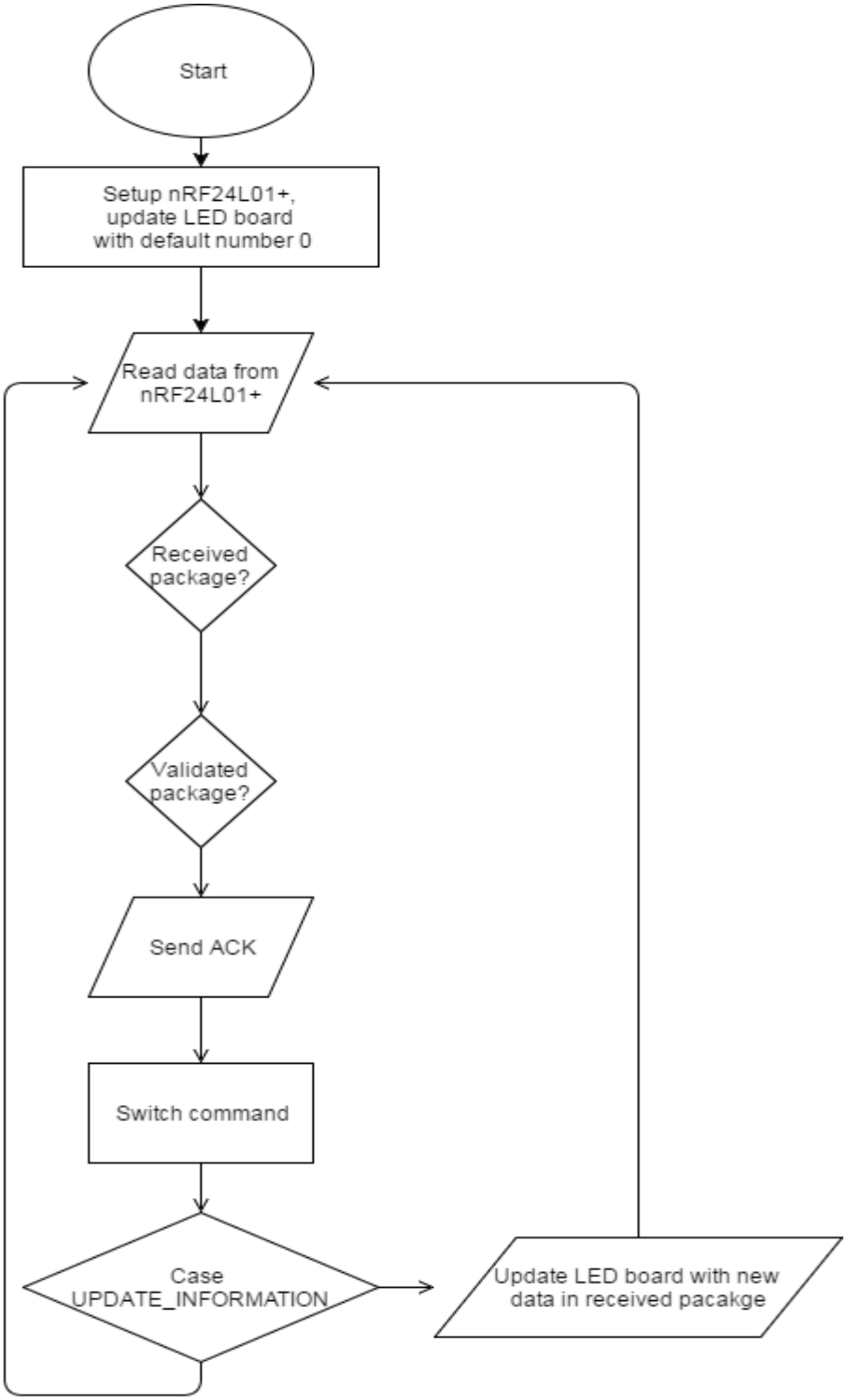


Figure 63: Interaction diagram - Sign node

4.3.3.3. Hub polling process

Summary: This diagram shows the process of the hub performs polling to get new data from lot nodes, update sign nodes if necessary, send updated information to API server and PubNub server

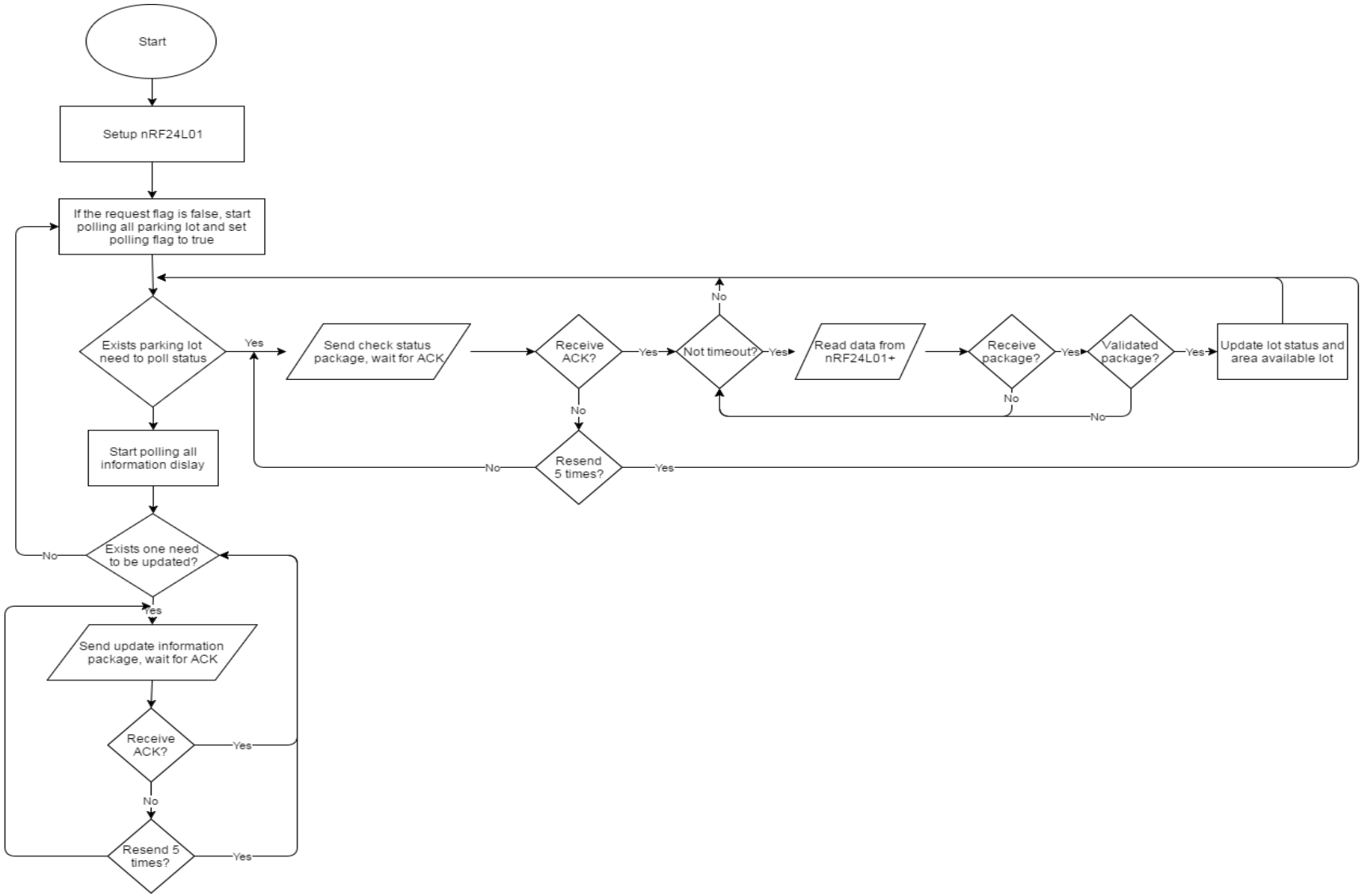


Figure 64: Interaction diagram - Hub polling process

4.3.3.1. Hub execute request process

Summary: This diagram shows the process of the hub receives requests from PubNub server and execute them.

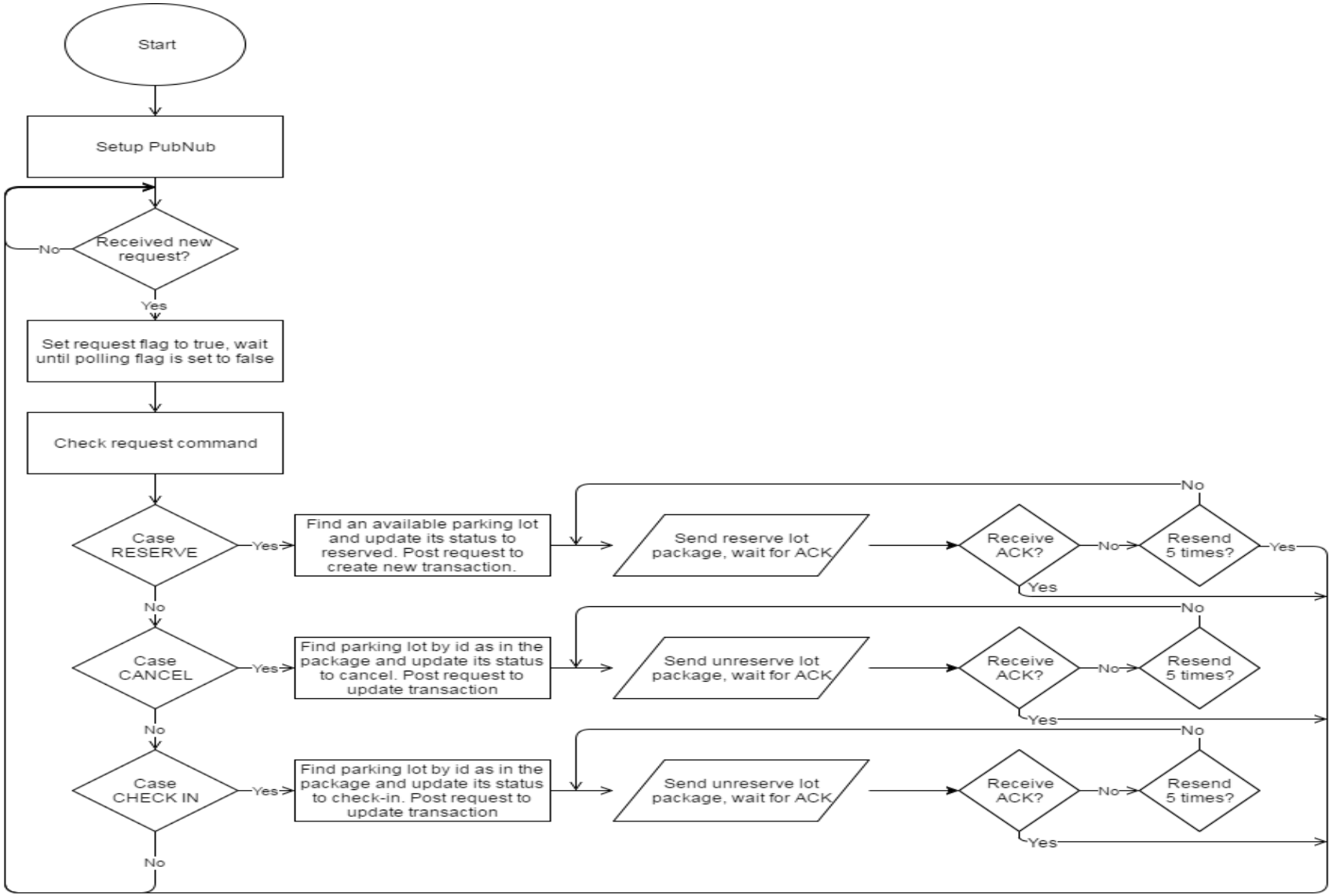


Figure 65: Interaction diagram - Hub request process

4.4. Hardware Detailed Description

4.4.1. Raspberry Pi 3 model B

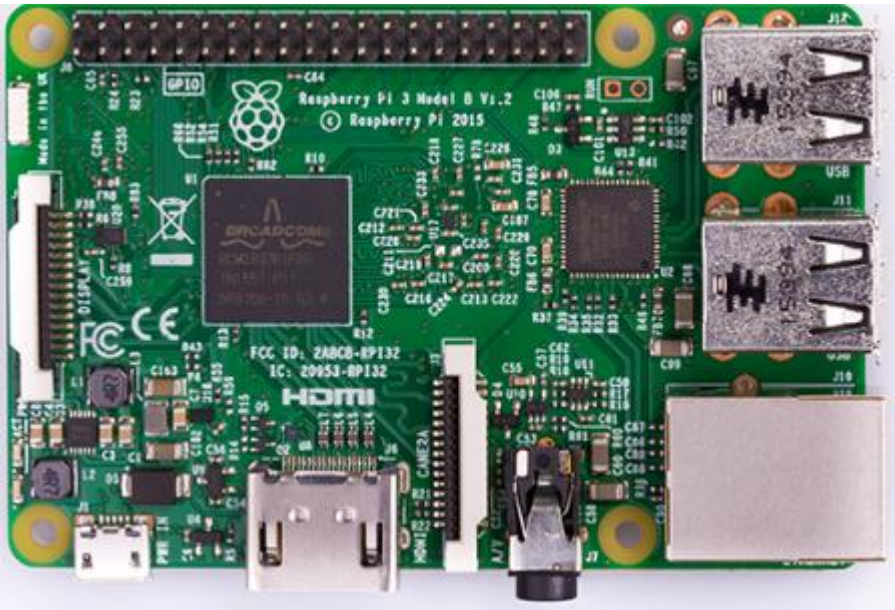


Figure 66: Raspberry Pi 3 model B

The Raspberry Pi 3 is the third generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016.

The Raspberry Pi 3 is a low cost, credit-card size computer that includes 802.11n Wi-Fi and Bluetooth 4.1. Even though it requires lots of additional hardware to function as a full PC with limited operating system selection, with the introduction of wireless connectivity and a boost in performance over its previous iteration, make the Raspberry Pi 3 Model B appropriate for a wider variety of projects – and it still costs just $35.

For our project, the Raspberry Pi 3 will be use as the Hub for IoT architecture. It will perform as a Gateway to connect the system to cloud server. The Hub will need to polling the information from Lot node continuously while waiting for User/Manager request on the cloud so the power to run multi-thread, coupled with build-in Wi-Fi and low cost is the main reason we use this device.

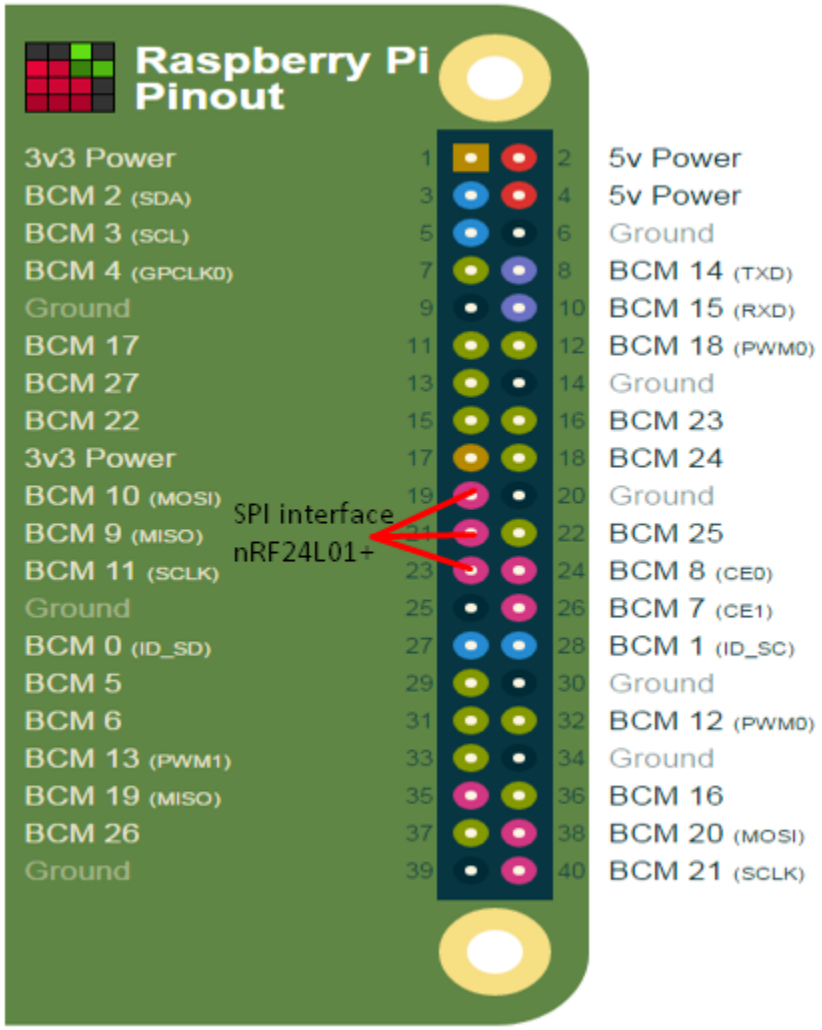


Figure 67: Raspberry Pi 3 Model B pinount

On the Raspberry Pi 3 we use Python 3 to build program. We use build-in libraries such as: GPIO, time, system… for information about our custom libraries and modules, please refer to previous section 4.3.

4.4.2. Arduino Nano

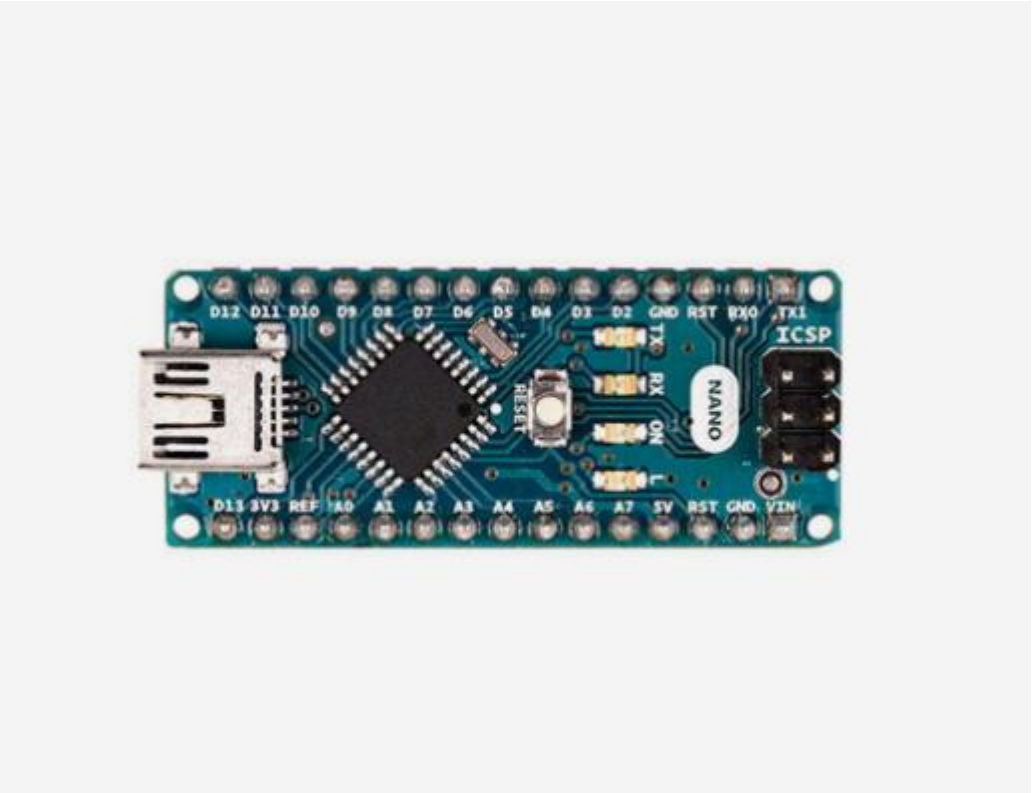


Figure 68: Arduino Nano

Arduino Nano is a surface mount breadboard embedded version with integrated USB. It is a small, complete, and breadboard-friendly board based on ATmega328 (Arduino Nano 3.x).

POWER:

The Arduino Nano can be powered via the mini-B USB connection, 6-20V unregulated external power supply (pin 30), or 5V regulated external power supply (pin 27). The power source is automatically selected to the highest voltage source.

PINOUT:

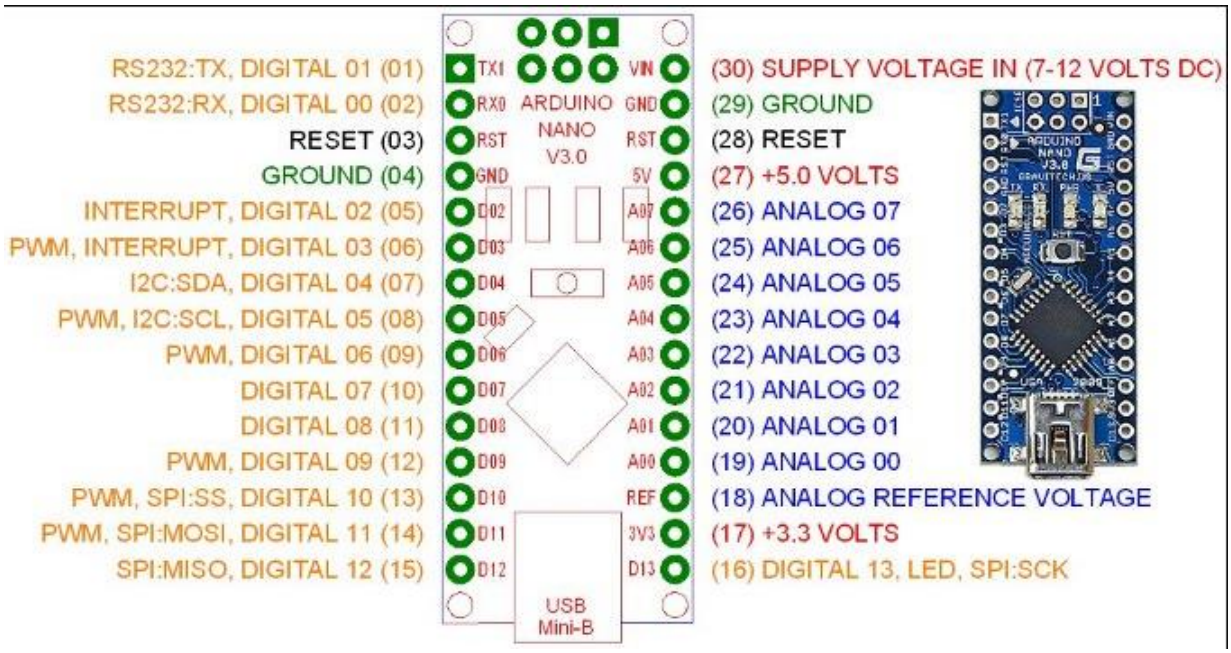


Figure 69: Arduino Nano pinout

In each Lot node and Sign node, we need a MCU to handling information from sensor, RF module and control other actuators. Arduino Nano is perfect for our need because of its power to run code, small size, low cost and easy to implement.

On Arduino Nano we use build-in libraries for I^2C, SPI interface and our custom libraries and modules. For more information about our custom libraries and modules, please refer to previous section 4.3.

4.4.3. Compass module 3-Axis HMC5883L

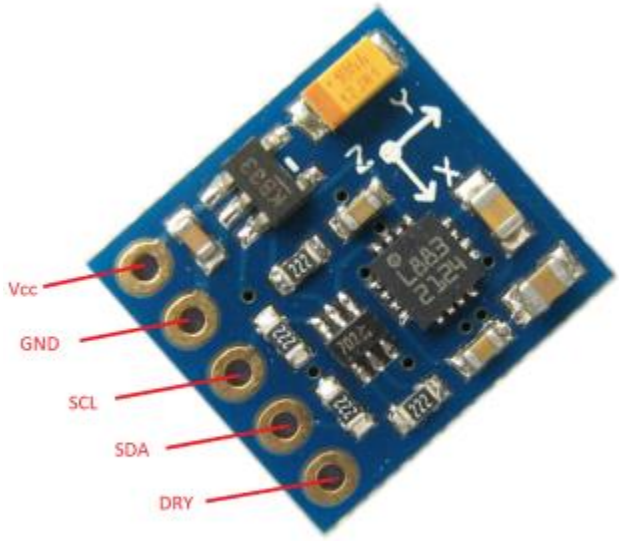


Figure 70: 3-Axis HMC5883L

The Honeywell HMC5883L is a surface-mount, multi-chip module designed for low-field magnetic sensing with a digital interface for applications such as low cost compassing and magnetometry.

We will place this sensor right at the middle of parking lot so it will detect if a car is parked in that lot or not. In theory, this magnetometer can detect the distorted flux line of earth's field created when it being cut by a large metallic body (ferrous) of the car. We have proved this test in a separated report (Test HMC5883L.xlxs) that the module provides noticeable offset, in other words, the car can be detected.

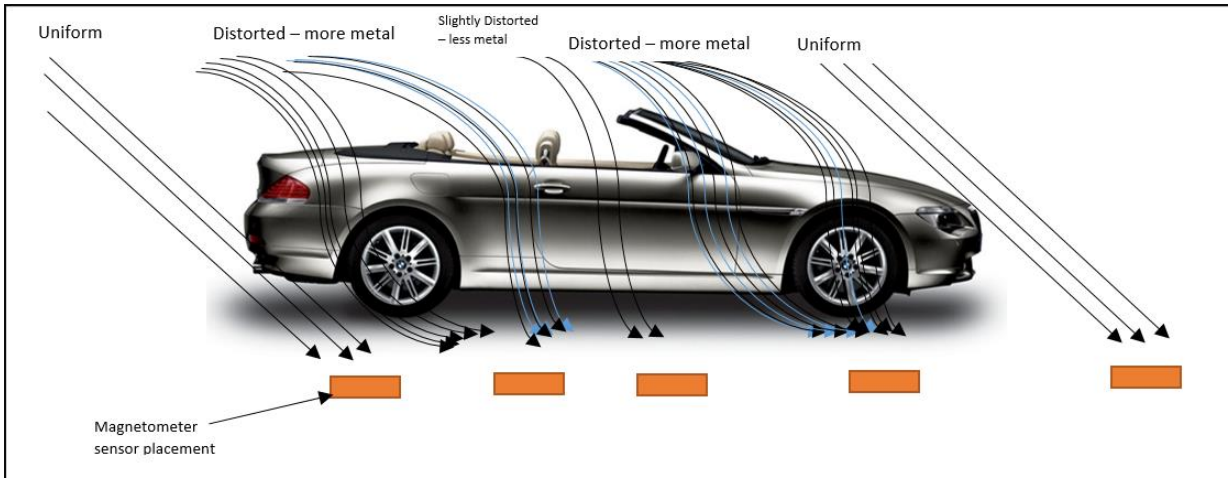


Figure 71: Earth's magnetic field lines distorted by ferrous metal of the car

PINOUT:

- **VIN:** this is the power in pin, give it 3-5VDC. It is reverse-polarity protected. Use the same voltage as you do for the controlling microcontroller's logic.
- **GND:** this is the common power and data ground pin.
- **SDA** and **SCL:** these are the I²C data and clock pins used to send and receive data from the module to your microcontroller. There are 10K pull-ups on these pins to the VIN pin. You can connect these pins to 5V I²C lines, there are level shifters on board to safely bring the pins down to 3V.
- **RDY:** this is the 'data ready' pin output. If you want to stream data at high speed (higher than 100 times a second) you may want to listen to this pin for when data is ready to be read.

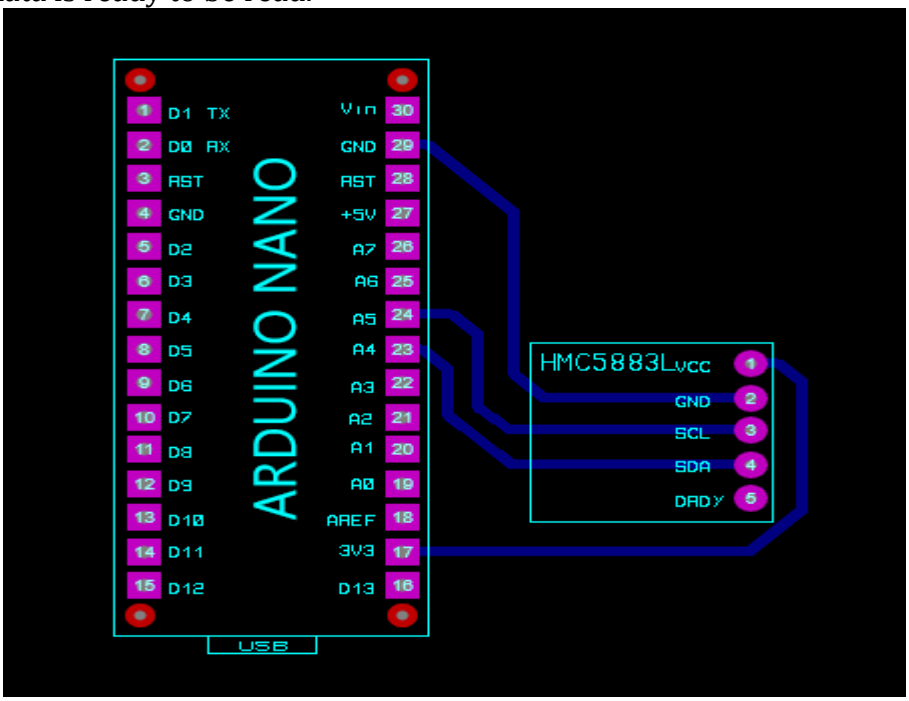


Figure 72: Arduino Nano and HMC5883L connection

ANISOTROPIC MAGNETO-RESISTIVE SENSORS

The Honeywell HMC5883L magneto-resistive sensor circuit is a trio of sensors and application specific support circuits to measure magnetic fields. With power supply applied, the sensor converts any incident magnetic field in the sensitive axis directions to a differential voltage output. The magneto-resistive sensors are made of a nickel-iron (permalloy) thin-film and patterned as a resistive strip element. In the presence of a magnetic field, a change in the bridge resistive elements causes a corresponding change in voltage across the bridge outputs.

These resistive elements are aligned together to have a common sensitive axis (indicated by arrows in the pinout diagram) that will provide positive voltage change with magnetic fields increasing in the sensitive direction. Because the output is only proportional to the magnetic field component along its axis, additional sensor bridges are placed at orthogonal directions to permit accurate measurement of magnetic field in any orientation.

MODES OF OPERATION

- **Continuous-Measurement Mode:** the device continuously makes measurements, at user selectable rate, and places measured data in data output registers.
- **Single-Measurement Mode:** this is the default power-up mode. During single-measurement mode, the device makes a single measurement and places the measured data in data output registers. After the measurement is complete and output data registers are updated, the device is placed in idle mode.
- **Idle Mode:** during this mode the device is accessible through the I^2C bus, but major sources of power consumption are disabled, such as, but not limited to, the ADC, the amplifier, and the sensor bias current. All registers maintain values while in idle mode. The I^2C bus is enabled for use by other devices on the network while in idle mode.

REGISTER LIST

Address Location	Name	Access
00	Configuration Register A	Read/Write
01	Configuration Register B	Read/Write
02	Mode Register	Read/Write
03	Data Output X MSB Register	Read
04	Data Output X LSB Register	Read
05	Data Output Z MSB Register	Read
06	Data Output Z LSB Register	Read
07	Data Output Y MSB Register	Read
08	Data Output Y LSB Register	Read
09	Status Register	Read
10	Identification Register A	Read
11	Identification Register B	Read
12	Identification Register C	Read

Table 191: HMC55883L Register list

IMPLEMENT:

We implemented the usage of this magnetometer in C++ code in a module name HMC5883L, which can be refer in previous section 4.3 for more information.

4.4.4. RF module nRF24L01+

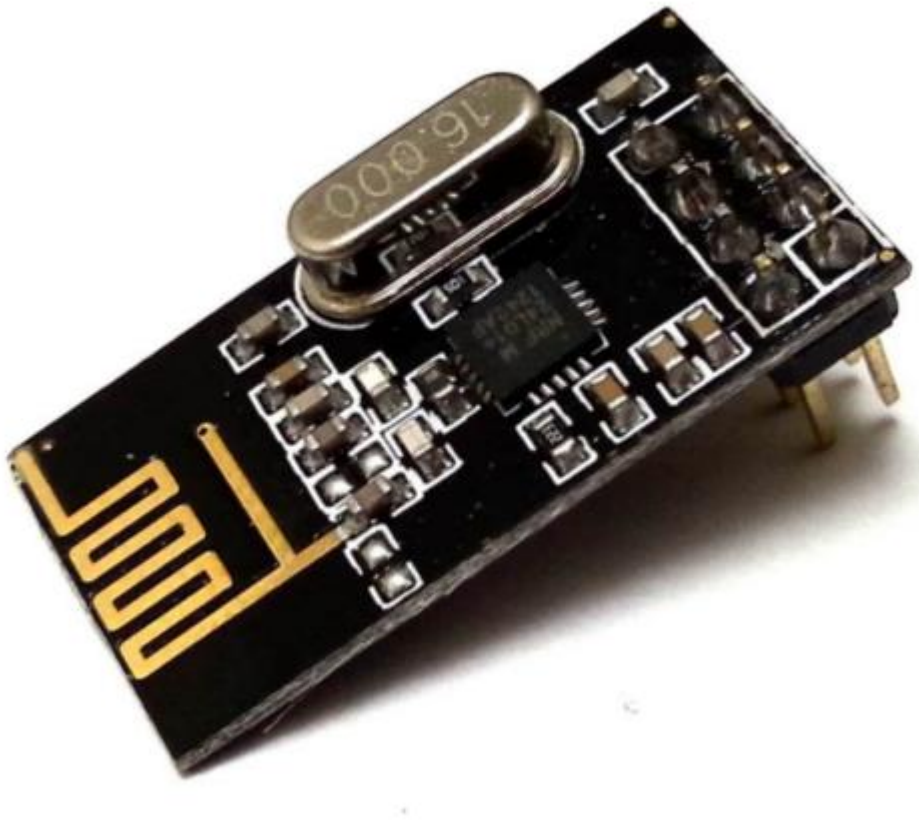


Figure 73: RF module nRF24L01+

The nRF24L01+ is a single chip 2.4GHz transceiver with an embedded baseband protocol engine (Enhanced ShockBurst™), suitable for ultra-low power wireless applications. The nRF24L01+ is designed for operation in the world wide ISM frequency band at 2.400 - 2.4835GHz.

This module will be used as the way for nodes and hub communicate with each other in our system. A car park is a very large area, meaning it have little to none obstructed object, so with a 2.4 GHz RF module, the system will have a longer range and a smaller antenna, meaning smaller size. This module also provides lots of other benefit such as multiple reading pipes, large payload packet… so it easy for us to implement a more complicated network model such as Tree, Mesh, Hop…

PINOUT:

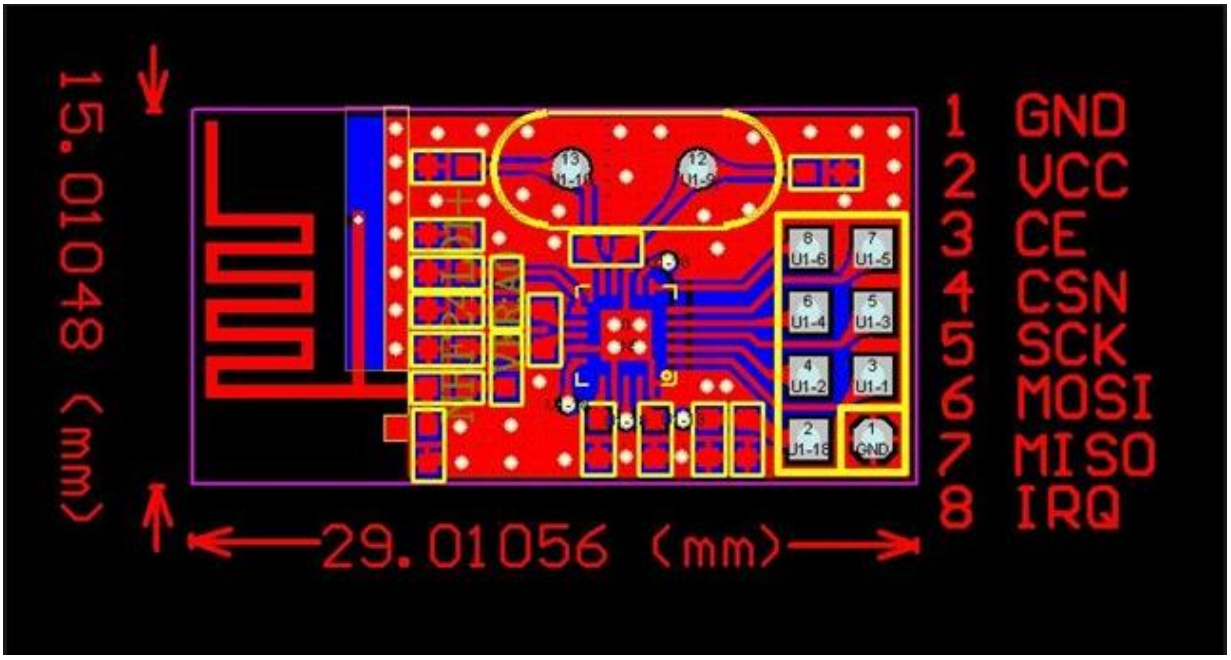


Figure 74: nRF24L01+ pinout

Pin	Name	Pin function	Description
1	GND	Power	Ground (0V)
2	VCC	Power	Power Supply (+1.9V - +3.6V DC)
3	CE	Digital Input	Chip Enable Actives RX or TX mode
4	CSN	Digital Input	SPI Chip Select
5	SCK	Digital Input	SPI Clock
6	MOSI	Digital Input	SPI Slave Data Input
7	MISO	Digital Output	SPI Slave Data Output, with tri-state option
8	IRQ	Digital Output	Maskable interrupt pin. Active low.

Table 192: nRF24L01+ pinout

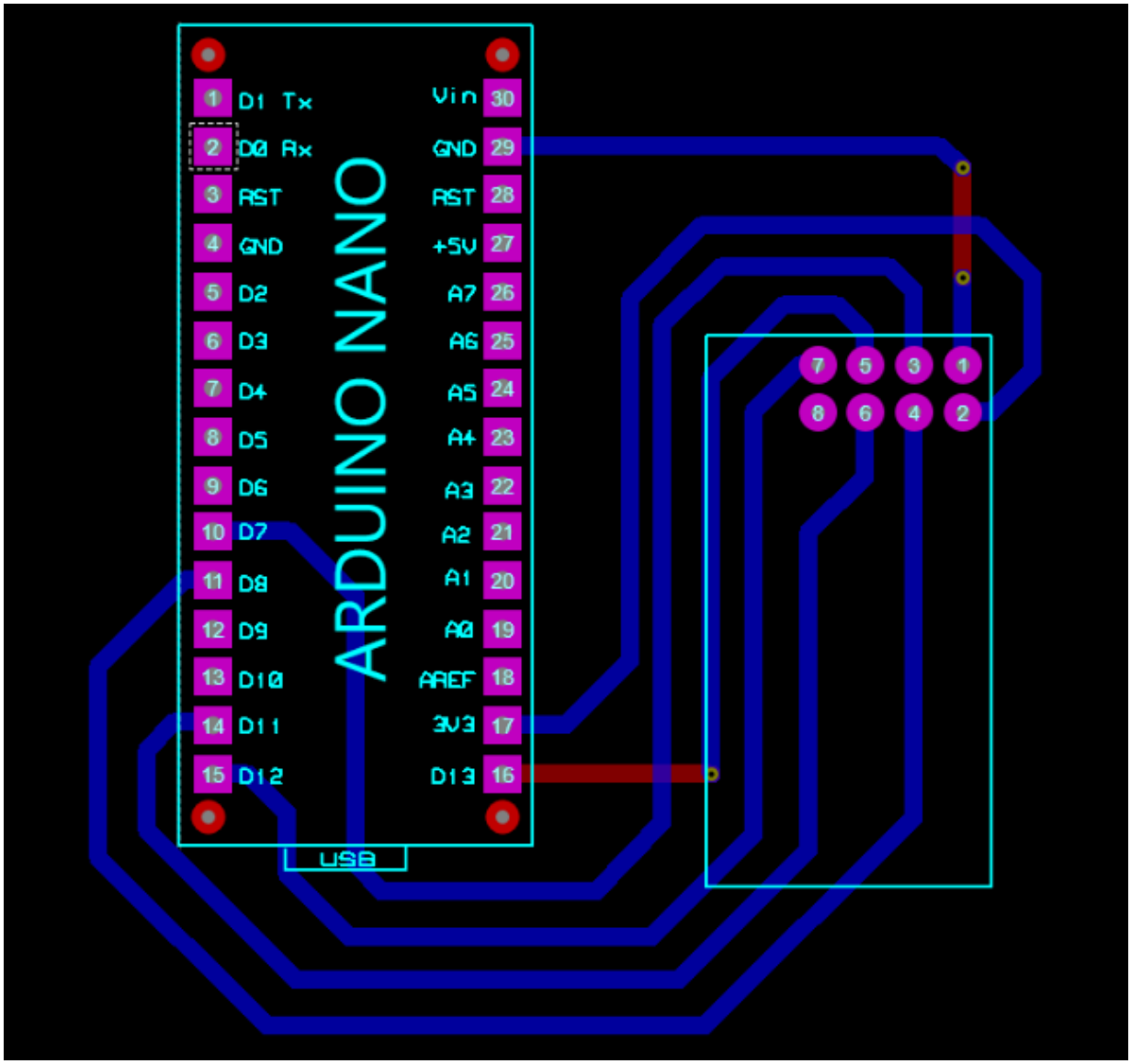


Figure 75: Arduino Nano and nRF24L01+ connection

MODES OF OPERATION

- **Power down mode:** In power down mode nRF24L01+ is disabled using minimal current consumption. All register values available are maintained and the SPI is kept active, enabling change of configuration and the uploading/downloading of data registers.
- **Standby-I mode:** By setting the PWR_UP bit in the CONFIG register to 1, the device enters standby-I mode. Standby-I mode is used to minimize average current consumption while maintaining short start up times. In this mode only part of the crystal oscillator is active. Change to active modes only happens if CE is set high and when CE is set low, the nRF24L01 returns to standby-I mode from both the TX and RX modes.
- **Standby-II mode:** In standby-II mode extra clock buffers are active and more

current is used compared to standby-I mode. nRF24L01+ enters standby-II mode if CE is held high on a PTX device with an empty TX FIFO. If a new packet is uploaded to the TX FIFO, the PLL immediately starts and the packet is transmitted after the normal PLL settling delay (130µs).

- **RX mode:** The RX mode is an active mode where the nRF24L01+ radio is used as a receiver. To enter this mode, the nRF24L01+ must have the PWR_UP bit, PRIM_RX bit and the CE pin set high. In RX mode the receiver demodulates the signals from the RF channel, constantly presenting the demodulated data to the baseband protocol engine. The baseband protocol engine constantly searches for a valid packet. If a valid packet is found (by a matching address and a valid CRC) the payload of the packet is presented in a vacant slot in the RX FIFOs. If the RX FIFOs are full, the received packet is discarded.
- **TX mode:** The TX mode is an active mode for transmitting packets. To enter this mode, the nRF24L01+ must have the PWR_UP bit set high, PRIM_RX bit set low, a payload in the TX FIFO and a high pulse on the CE for more than 10µs. The nRF24L01+ stays in TX mode until it finishes transmitting a packet.

PACKET FORMAT:

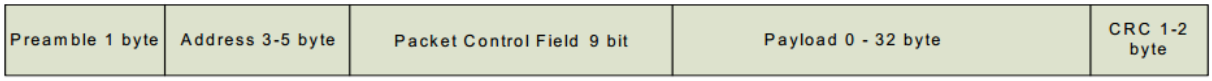


Figure 76: nRF24L01+ packet format

- **Preamble:** The preamble is a bit sequence used to synchronize the receiver's demodulator to the incoming bit stream. The preamble is one byte long and is either 01010101 or 10101010. If the first bit in the address is 1 the preamble is automatically set to 10101010 and if the first bit is 0 the preamble is automatically set to 01010101. This is done to ensure there are enough transitions in the preamble to stabilize the receiver.
- **Address:** This is the address for the receiver. An address ensures that the packet is detected and received by the correct receiver, preventing accidental cross talk between multiple nRF24L01+ systems.
- **Packet Control Field:** The packet control field contains a 6 bit **payload length** field, a 2 bit **PID** (Packet Identity) field and a 1 bit **NO_ACK flag**. **Payload length** specifies the length of the payload in bytes. **PID** is used to detect if the received packet is new or retransmitted. **NO_ACK flag** is only used when the auto acknowledgement feature is used.
- **Payload:** The payload is the user defined content of the packet. It can be 0 to 32 bytes wide and is transmitted on-air when it is uploaded to nRF24L01+.
- **CRC:** The CRC is the mandatory error detection mechanism in the packet. It is either 1 or 2 bytes and is calculated over the address, Packet Control Field and Payload.

SPI COMMANDS

Command name	**Command**	**# Data bytes**	**Operation**
R_REGISTER	000A AAAA	1 to 5 LSByte first	Read command and status registers. AAAAA = 5 bit Register Map Address
W_REGISTER	001A AAAA	1 to 5 LSByte first	Write command and status registers. AAAAA = 5 bit Register Map Address Executable in power down or standby modes only.
R_RX_PAYLOAD	0110 0001	1 to 32 LSByte first	Read RX-payload: 1 – 32 bytes. A read operation always starts at byte 0. Payload is deleted from FIFO after it is read. Used in RX mode.
W_TX_PAYLOAD	1010 0000	1 to 32 LSByte first	Write TX-payload: 1 – 32 bytes. A write operation always starts at byte 0 used in TX payload.
FLUSH_TX	1110 0001	0	Flush TX FIFO, used in TX mode
FLUSH_RX	1110 0010	0	Flush RX FIFO, used in RX mode Should not be executed during transmission of acknowledge, that is, acknowledge package will not be completed.
REUSE_TX_PL	1110 0011	0	Used for a PTX device Reuse last transmitted payload. TX payload reuse is active until W_TX_PAYLOAD or FLUSH TX is executed. TX payload reuse must not be activated or deactivated during package transmission.
R_RX_PL_WIDa	0110 0000	1	Read RX payload width for the top R_RX_PAYLOAD in the RX FIFO.
W_ACK_PAYLOADa	1010 1PPP	1 to 32 LSByte first	Used in RX mode. Write Payload to be transmitted together with ACK packet on PIPE PPP. (PPP valid in the range from 000 to 101). Maximum three ACK packet payloads can be pending. Payloads with same PPP are handled using first in - first out principle. Write payload: 1– 32 bytes. A write operation always starts at byte 0.

W_TX_PAYLOAD_NOACKa	1011 0000	1 to 32 LSByte first	Used in TX mode. Disables AUTOACK on this specific packet.
NOP	1111 1111	0	No Operation. Might be used to read the STATUS register

Table 193: nRF24L01+ SPI commands

IMPLEMENT:

We implemented the usage of this magnetometer in C++ code in a module name rf_meta, which can be refer in previous section 4.3 for more information.

4.4.5. Lot node

4.4.5.1. RGB LED common anode

Figure 77: RGB LED common anode

RGB LED allows you to change the lights to any color to show state of parking slot.

In a real product environment, we will use led strip, but for current scope, in demo environment, we use a single RGB LED, which work in the same way as led strip, as an indicator for lot in the car park. The led will change color depended on the current status of the lot, for example, red if the lot is occupied and green if the lot is emptied.

IMPLEMENT:

We implemented the usage of this magnetometer in C++ code in a module name RF_META, which can be refer in previous section 4.3 for more information.

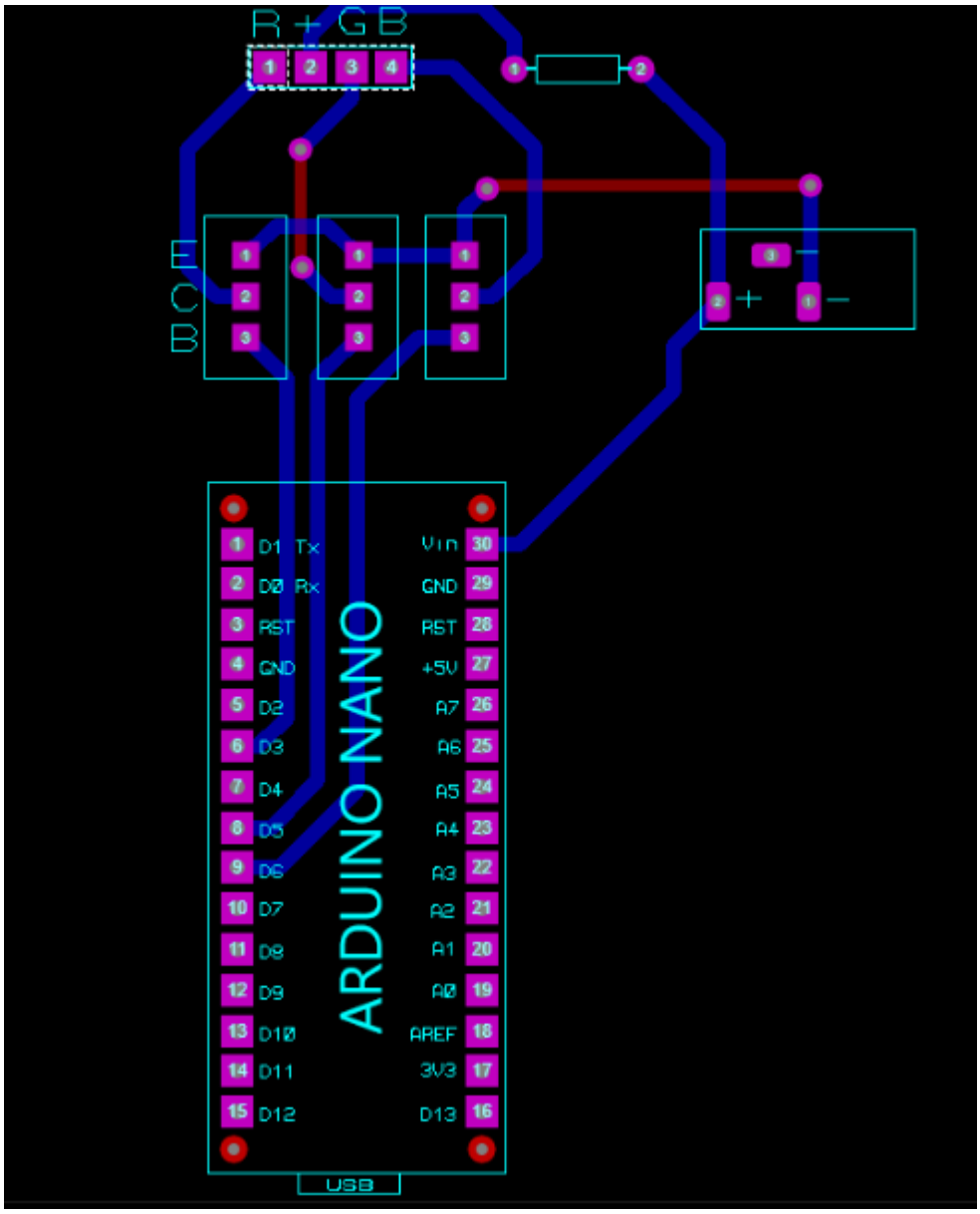


Figure 78: Arduino Nano and RGB LED connection through TIP122

4.4.5.2. Transistor TIP122

Figure 79: Transistor TIP122

A single digital pin on Arduino Nano do not provide enough current to power RGB LED, A solution for this situation is to use an NPN Darlington Transistor designed for medium power linear switching applications, so we use TIP122 Transistor to provide RGB LED with power from an external source. It can power devices up to 100VDC at 5 Amps.

PINOUT:

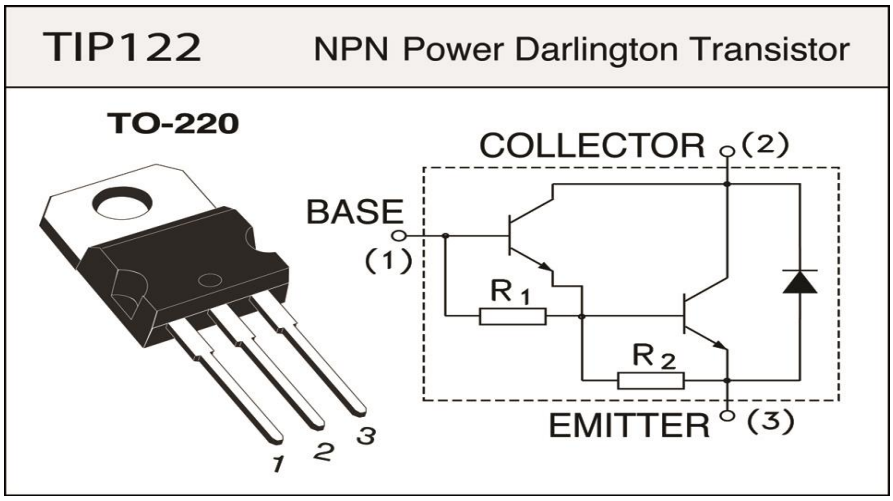


Figure 80: Transistor TIP122 pinout

4.4.5.3. Servo motor SG90

Figure 81: Servo motor SG90

In demo environment, we use the servo motor SG90 to control the model barrier in each parking lot. In default position, the servo will return to 0 value, and if the lot is reserved, or is disabled by manager, the servo will change value to 90 and the barrier will block any car try to enter that lot.

PINOUT:

Pin of Servo SG90	Name	Description
Red	VCC	Power supply 5V
Black	GND	Power supply ground
Yellow	Signal	The servo will move based on the signal sent to signal wire.

Table 194: Servo motor SG90 pinout

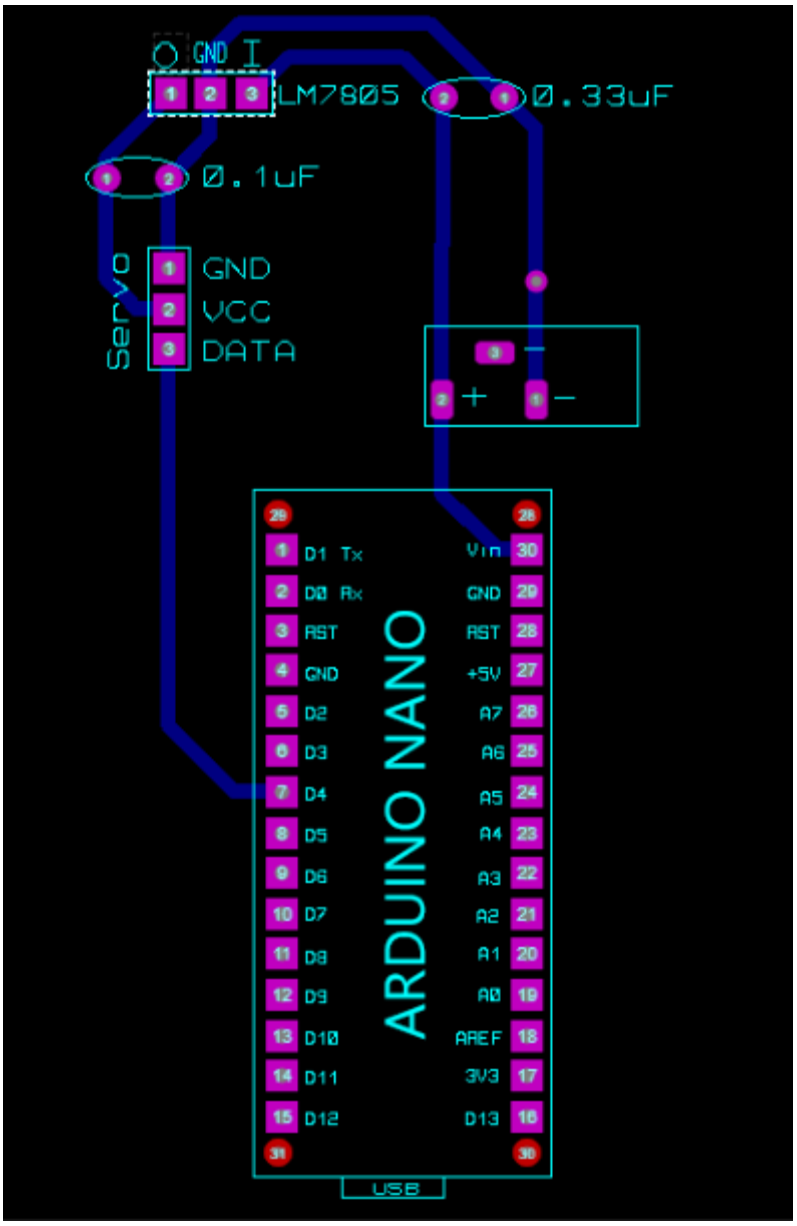


Figure 82: Arduino Nano and Servo SG90 connection, with LM7805

We connect this module to Arduino Nano with pin D4 and control it using the Servo library provided by Arduino.

4.4.5.4. LM7805

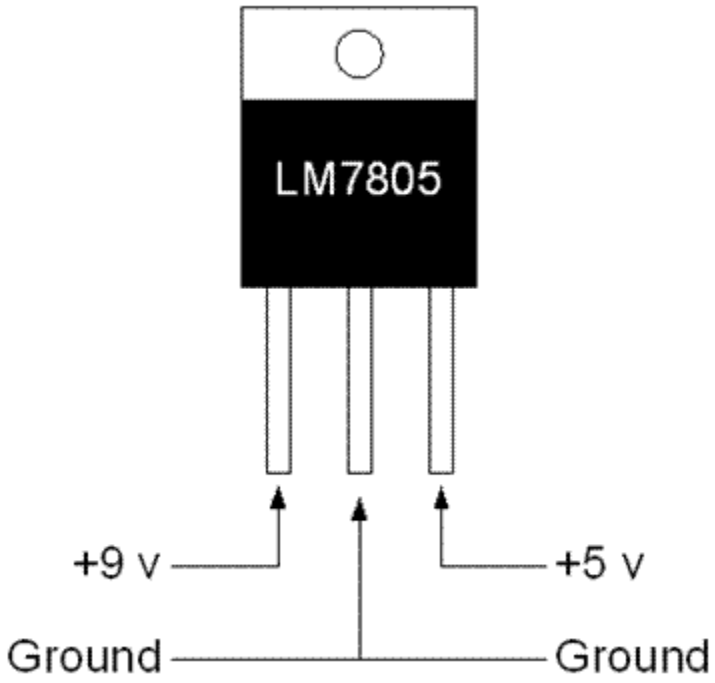


Figure 83: LM7805

Because the above servo SG90 operating voltage is 4.8V (~5V) so we need to connect it with this voltage regulator to make sure that the voltage provided from our Arduino Nano can be adjusted to eliminate noise and distribution problems associated with single-point regulation.

The connection of LM7805 with Arduino Nano can be seen in the above Figure 82.

4.4.6. Sign node

4.4.6.1. Power logic 8-bit shift register TPIC6B595

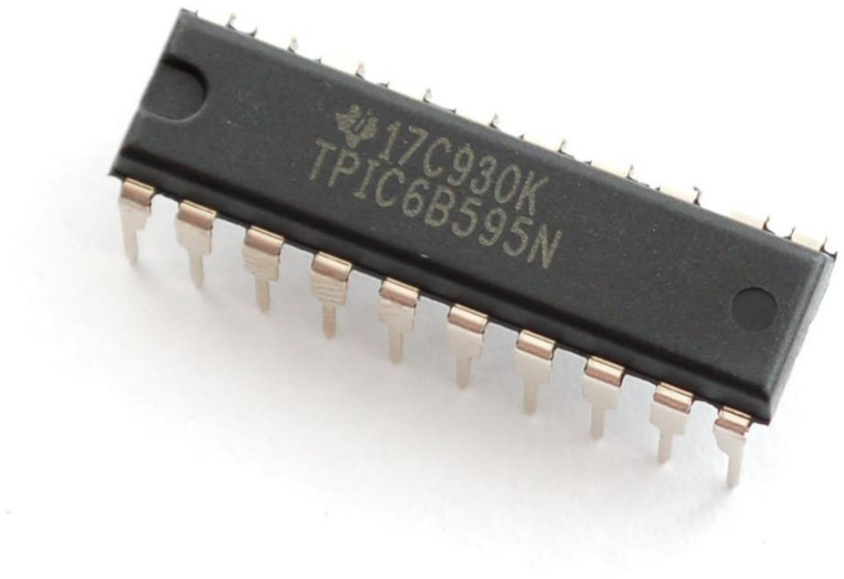


Figure 84: Power logic 8-bit shift register TPIC6B595

To display high power 7-segment display, we choose IC TPIC6B595 instead of IC 74HC595 because TPIC6B595 is a simple shift register IC that can control high-voltage/high-current devices directly**.** Each output pin can sink 150mA and then supports the maximum of Load Voltage at 50V.

PINOUT:

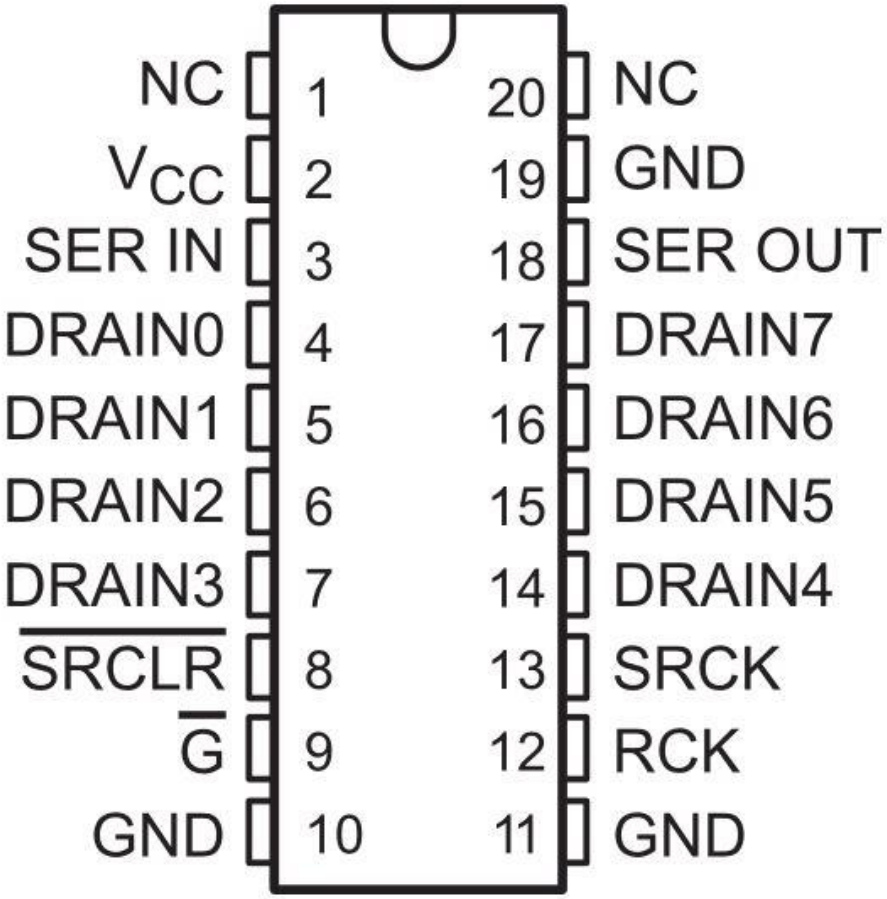


Figure 85: Power logic 8-bit shift register TPIC6B595 pinout

Pin		I/O	Description
Name	**No.**		
DRAIN0	4	O	Open-drain output
DRAIN1	5		
DRAIN2	6		
DRAIN3	7		
DRAIN4	14		
DRAIN5	15		
DRAIN6	16		
DRAIN7	17		
G	9	I	Output enable, active-low

GND	10,11,19	-	Power ground
NC	1, 20	-	No internal connection
RCK	12	I	Register clock
SERIN	3	I	Serial data input
SEROUT	18	O	Serial data output
SRCK	15	I	Shift register clock
SRCLR	8	I	Shift register clear, active-low
VCC	2	I	Power supply

Table 195: Power logic 8-bit shift register TPIC6B595 pinout

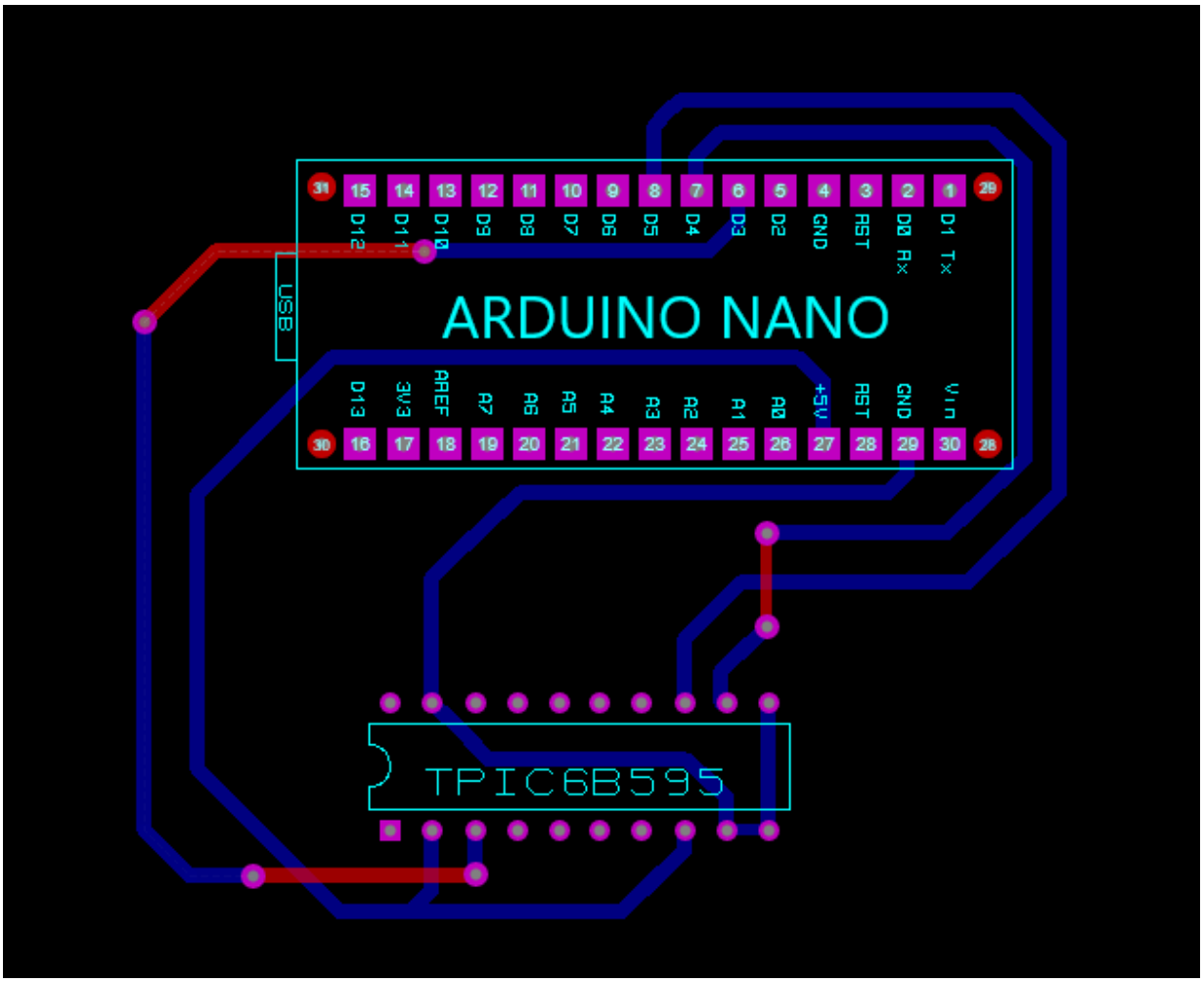


Figure 86: Arduino Nano and TPIC6B595 connection

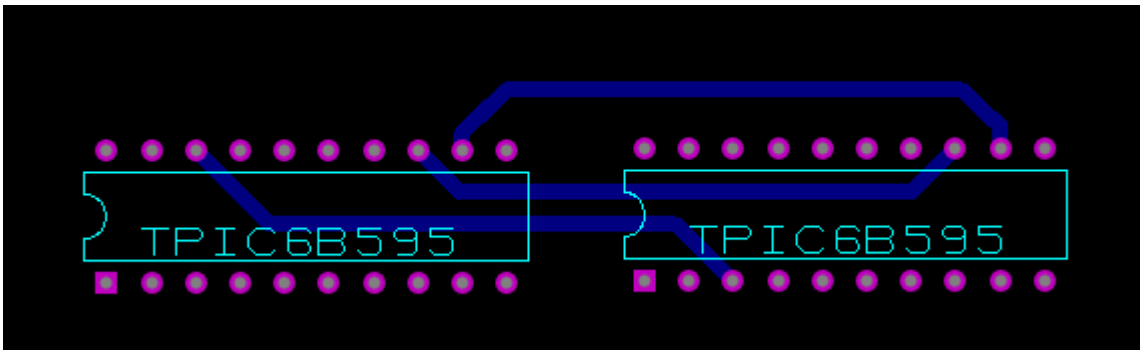


Figure 87: Sequence connection between 2 TPIC6B595

4.4.6.2. 7-Segment LED display

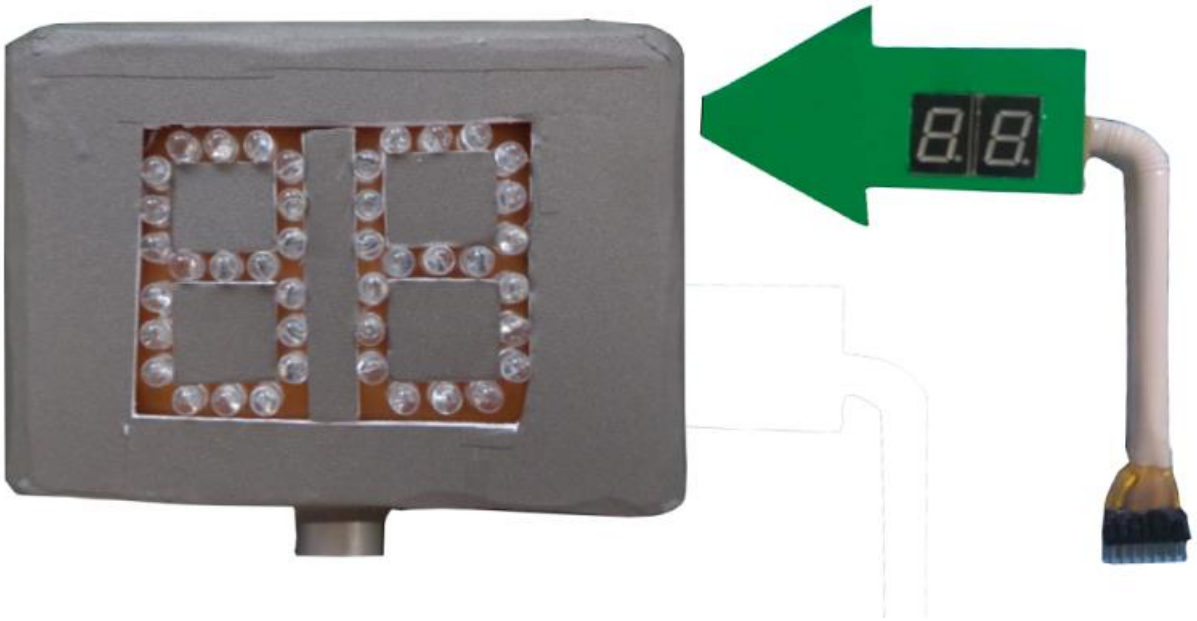


Figure 88: 7-Segment LED display

For demo sake, we use both rebuild 7-segment led and our custom created 7-segment led for sign nodes and main gate display board. All the 7-segment led used in our project is common anode.

The led in the sign nodes displays the available lots in their respected zone. The led in the main gate display board displays the total available lots of all the sign nodes.

The usage of these 7-segment led and the above shift register is implemented by us in a module name SEVEN_SEGMENT, which can be referred to in previous section 4.3 for more information.

4.4.7. Components connection

4.4.7.1. Overview component connection

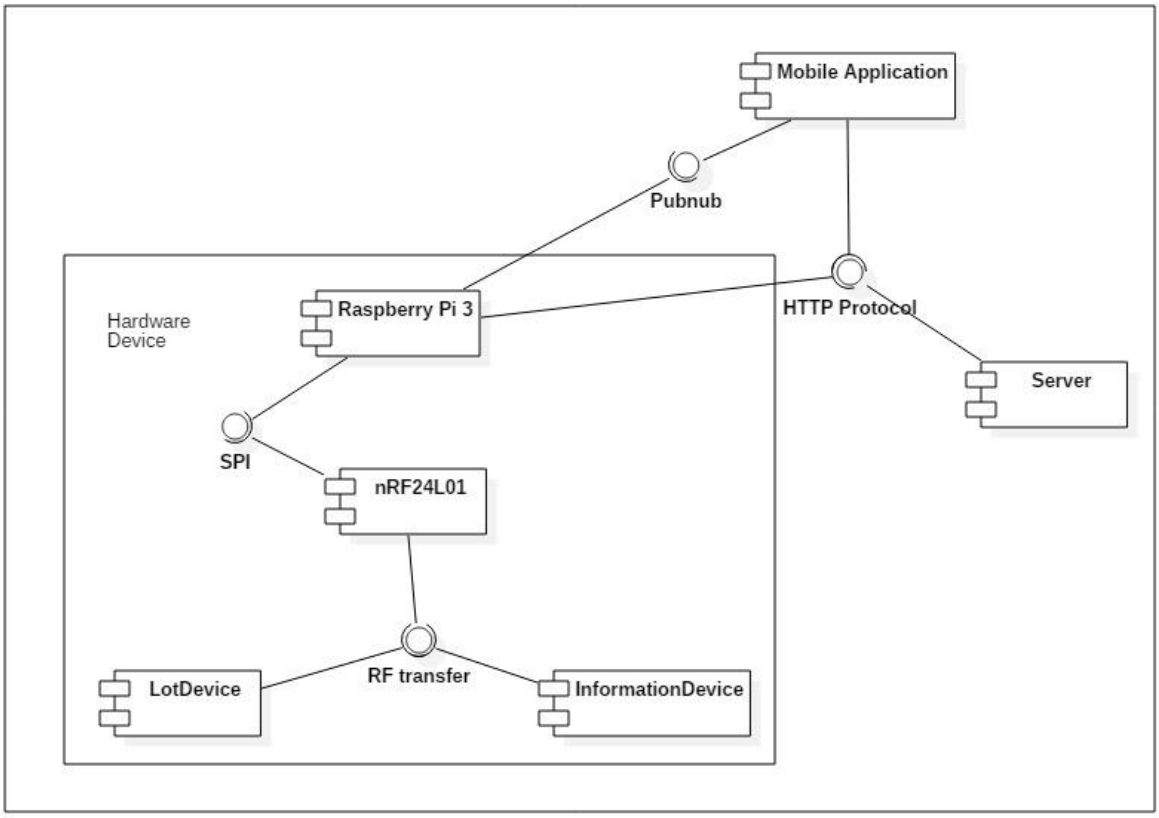


Figure 89: General component connection

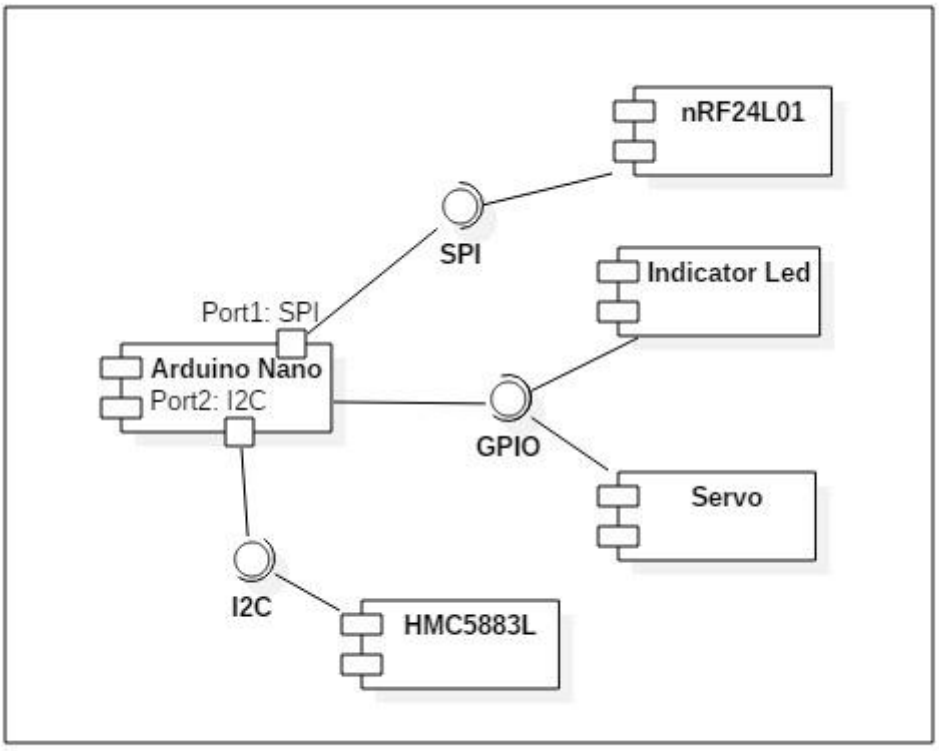


Figure 90: Lot node connection

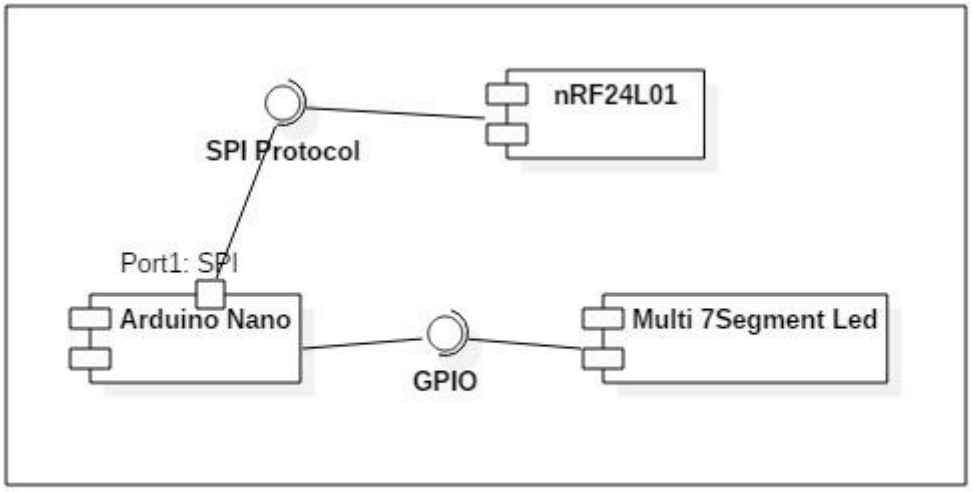


Figure 91: Sign node connection

4.4.7.2. Raspberry Pi 3 connection

Rpi 3	NRF24L01+
25 – Ground	1 – GND
17 – 3v3	2 – VCC
15 – GPIO22	3 – CE
24 – GPIO8	4 – CSN
23 – SCKL	5 – SCK
19 – MOSI	6 – MOSI
21 – MISO	7 - MISO

Table 196: Raspberry Pi 3 connection

4.4.7.3. Lot node connection

Arduino Nano	HMC5883L	NRF24L01+	RGB LED	Servo
6 – D3			Red pin (through TIP122)	
7 – D4				Data pin
8 – D5			Green pin (through TIP122)	
9 – D6			Blue pin (through TIP122)	
10 – D7		3 – CE		
11 – D8		4 – CS		
14 – D11		6 – MOSI		
15 – D12		7 – MISO		
16 – D13		5 - SCK		
23 – A4	4 – SDA			
24 – A5	3 – SCL			

Table 197: Lot node connection

4.4.7.4. Sign node connection

Arduino Nano	NRF24L01+	TPIC6B595
5V		2 – VCC, 8 – SRCLR
Ground		9, 10, 11 - GND
6 – D3		3 – SER_IN
7 – D4		12 – RCK
8 – D5		13 - SRCK
10 – D7	3 – CE	
11 – D8	4 – CS	
14 – D11	6 – MOSI	
15 – D15	7 – MISO	
16 – D13	5 - SCK	

Table 198: Sign node connection

TPIC6B595 (1)	TPIC6B595 (2)
12 – RCK	12 – RCK
13 – SRCK	13 – SRCK
18 – SER_OUT	3 – SER_IN

Table 199: TPIC6B595s sequence connection

5. Interface

5.1. Component Interface

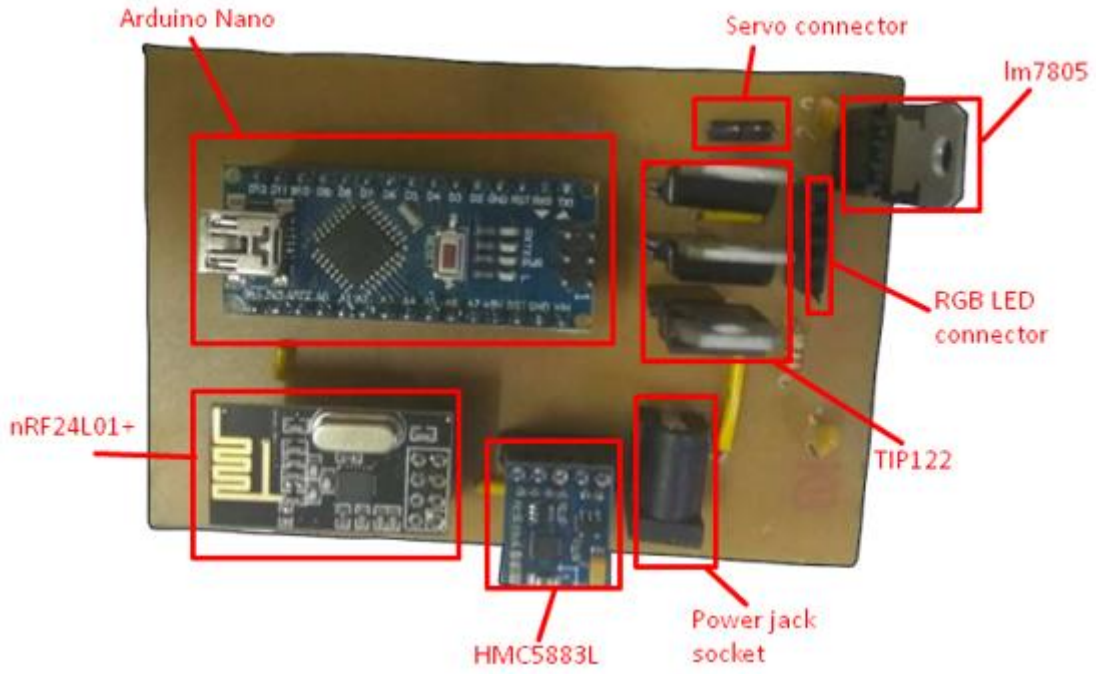


Figure 92: Lot node interface

This is the PCB of lot node. This PCB will be placed in the middle of parking lot so it can detect when a car is parked right above it. Because this PCB will be placed underground so the servo and RGB LED will connect with male and female header row respectively. In demo environment, we placed this directly under our car park model, but in product environment, this will be placed inside a protector.

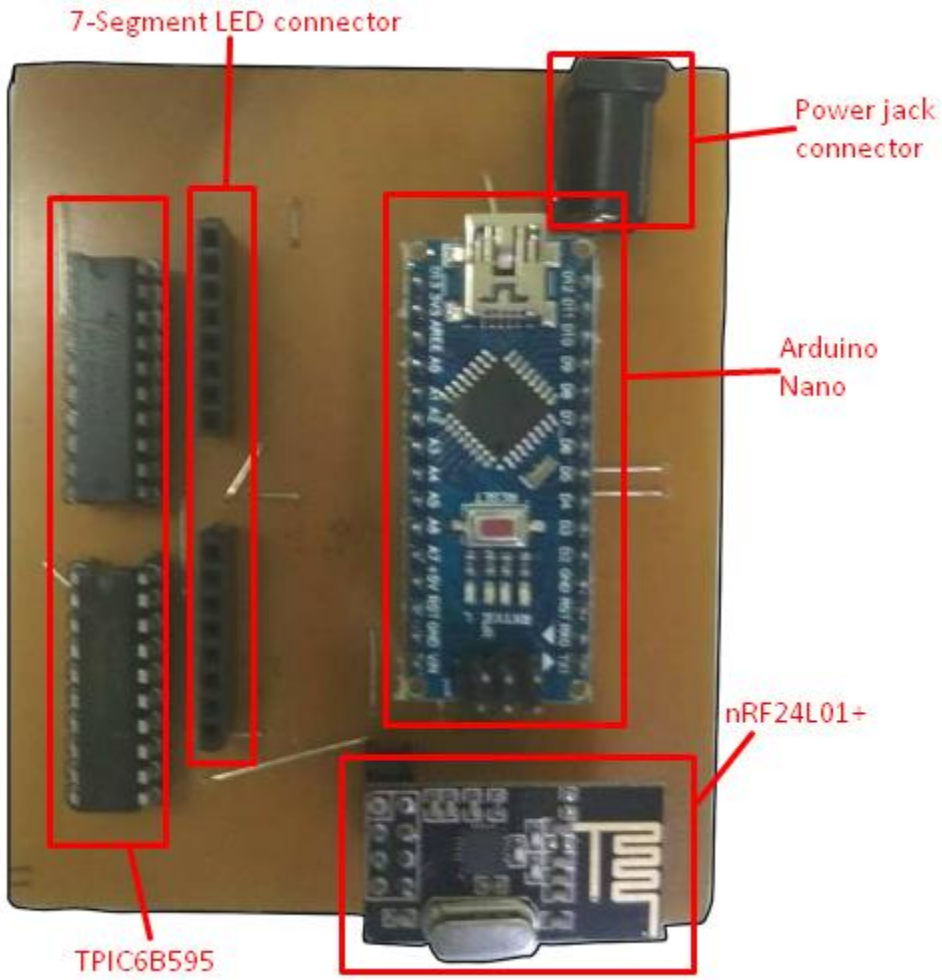


Figure 93: Sign node interface

This is the PCB of sign node. This PCB will connect to 7-segment led in our sign to display the number of available lots. In product environment, the placement of this PCB will be depended on its connected sign, but in demo environment, for simplicity's sake, we also placed this directly under our model, same as lot node.

5.2. User Interface Design

5.2.1. Common Interface

5.2.1.1. Login screen

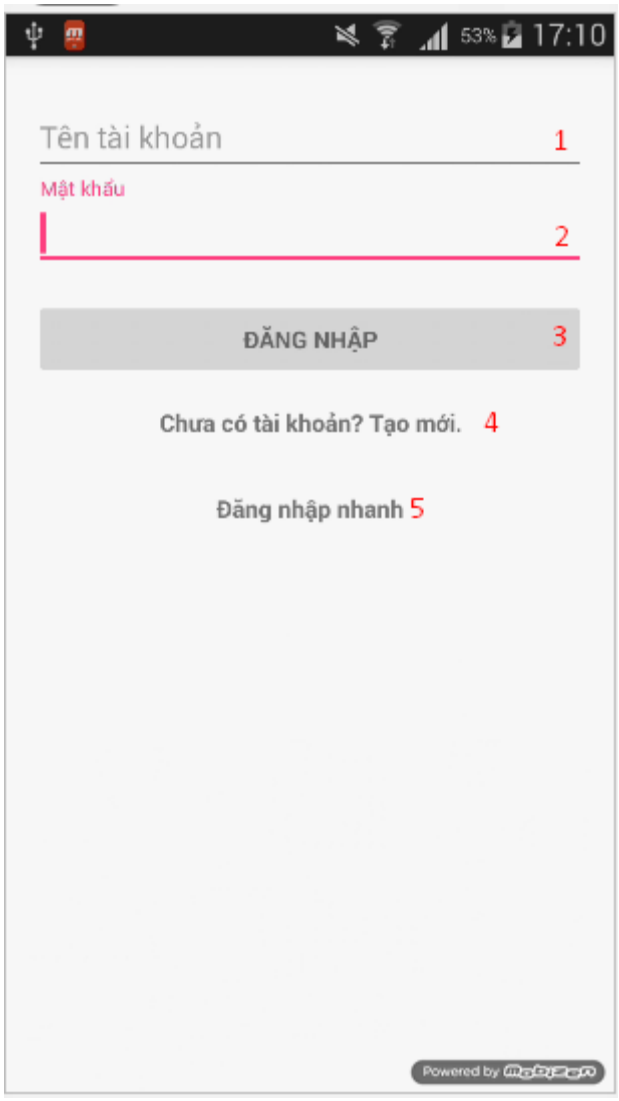


Figure 94: Login screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Username	Fill in username	N/A	Yes	Edittext	String	N/A
2	Password	Fill in password	N/A	Yes	Edittext	String	N/A

Table 200: Login screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
3	Sign in	Log-in into the system	N/A	Transfer to User Activity
4	Register	Register new account	N/A	Transfer to Register Activity
5	Sign In as Guest	Log-in into the system as Guest	N/A	Transfer to User Activity

Table 201: Login screen button/hyperlinks

5.2.1.2. Register screen

Figure 95: Register screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Email	Fill in email	N/A	Yes	Edittext	String	N/A
2	Username	Fill in username	N/A	Yes	Edittext	String	N/A
3	Fullname	Fill in fullname	N/A	Yes	Edittext	String	N/A
4	Password	Fill in password	N/A	Yes	Edittext	String	N/A
5	Confirm Password	Fill in confirm password	N/A	Yes	Edittext	String	N/A

Table 202: Register screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
6	Register	Register new account	N/A	Transfer to Login Activity
7	Sign up	Sign up to app	N/A	Transfer to Login Activity

Table 203: Register screen button/hyperlinks

5.2.2. User/Guest Interface

5.2.2.1. Map view screen

Figure 96: Map view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
2	Location Search	Fill in location name to search	N/A	No	Editttext	String	N/A
4	Current Location	The dot show current location of user	Yes	Yes	Image	Drawable	N/A

5	Car park marker	Map marker of car park	Yes	Yes	Image	Drawable	N/A

Table 204: Map view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
1	Navigation drawer	Open navigation drawer	N/A	Open navigation drawer
3	Current Location	Button to center map to current location	N/A	Center map to current location
6	Zoom in	Map zoom in button	N/A	Map zoom in
7	Zoom out	Map zoom out button	N/A	Map zoom out

Table 205: Map view screen button/hyperlinks

5.2.2.2. Navigation view

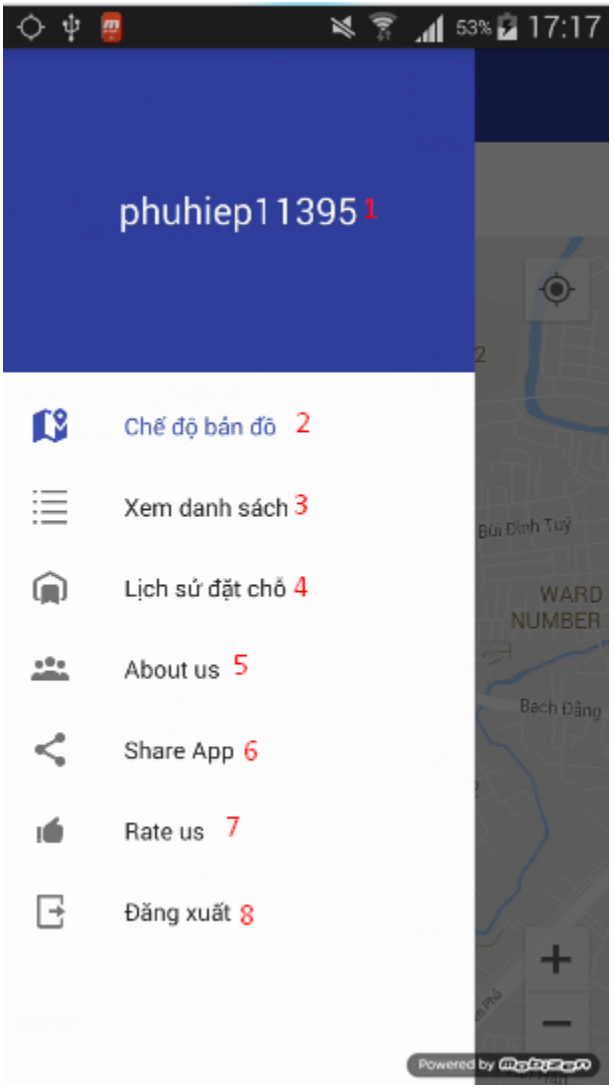


Figure 97: Navigation view

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Username	Fill in username	Yes	Yes	Textview	String	N/A

Table 206: Navigation view fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
2	Map view	Change to map view button	N/A	Transfer to User Activity

3	List view	Change to car park list view button	N/A	Transfer to Car Park List Activity
4	Transaction view	Change to history transaction list view button	N/A	Transfer to Transaction Activity
5	About us	About us button	N/A	Show about us dialog
6	Share app	Share app button	N/A	Share app on social network
7	Rate us	Rate us button	N/A	Transfer to app download page on app store
8	Logout	Logout button	N/A	Transfer to Login Activity

Table 207: Navigation view button/hyperlinks

5.2.2.3. Car park list view screen

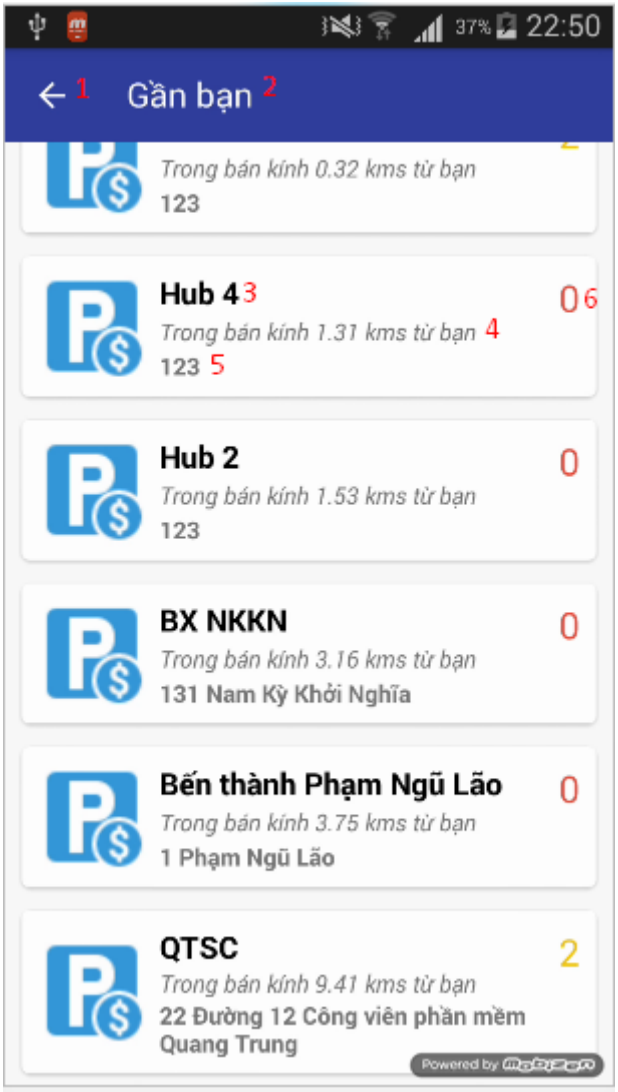


Figure 98: Car park list view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
2	Title	Display searched location or user location	Yes	Yes	Textview	String	N/A
3	Car park name	Name of car park	Yes	Yes	Textview	String	N/A
4	Distance away	Distance away from car park	Yes	Yes	Textview	String	N/A

5	Address	Address of car park	Yes	Yes	Textview	String	N/A
6	Available lot	Current available lot of car park	Yes	Yes	Textview	String	N/A

Table 208: Car park list view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
1	Back	Back to previous activity	N/A	Transfer to previous activity

Table 209: Car park list view screen button/hyperlinks

5.2.2.4. Car park detail view screen

Figure 99: Car park detail view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
2	Name	Name of car park	Yes	Yes	Textview	String	N/A
3	Distance away	Distance away from car park	Yes	Yes	Textview	String	N/A
4	Available lot	Current available lot of car park	Yes	Yes	Textview	String	N/A
8	Address	Address of car park	Yes	Yes	Textview	String	N/A
9	Phone	Phone of car park	Yes	Yes	Textview	String	N/A
10	Email	Email of car park	Yes	Yes	Textview	String	N/A
11	Description	Description of car park	Yes	Yes	Textview	String	N/A

Table 210: Car park detail view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
1	Back	Back to previous activity	N/A	Transfer to previous activity
5	Call	Call button	N/A	Call action using car park phone
6	Routing	Request routing from map app	N/A	Transfer to Map App with routing request
7	Reserve	Request reserve parking lot	N/A	Show reservation dialog

Table 211: Car park detail view screen button/hyperlinks

5.2.2.5. History transaction list view screen

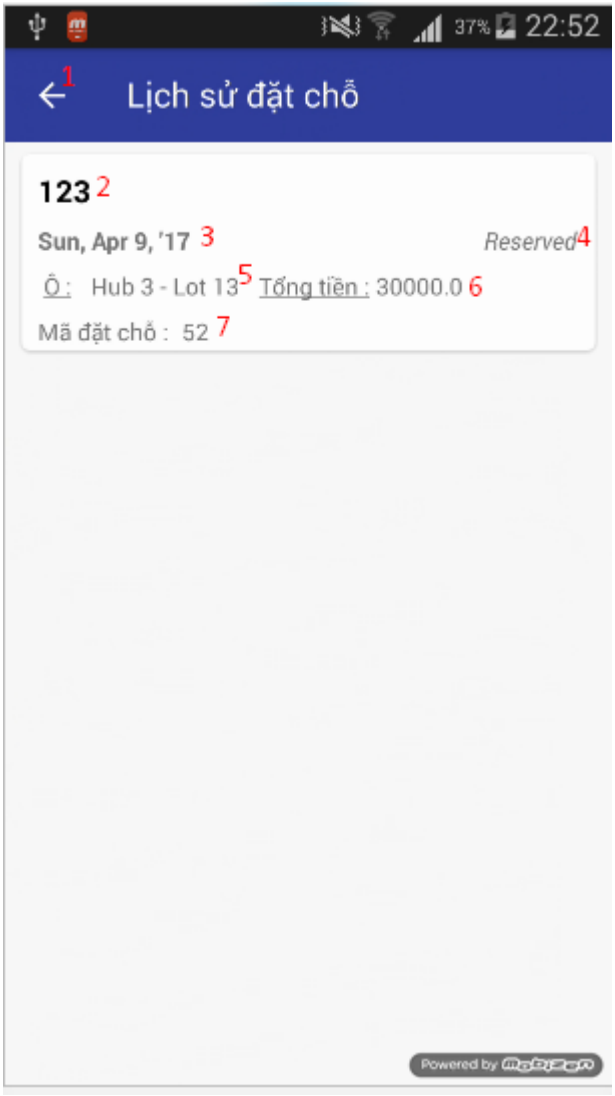


Figure 100: History transaction list view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
2	Address	Address of car park	Yes	Yes	Textview	String	N/A
3	Date	Date of reservation	Yes	Yes	Textview	String	N/A
4	Status	Status of transaction	Yes	Yes	Textview	String	N/A
5	Lot name	Name of reserved parking lot	Yes	Yes	Textview	String	N/A

6	Cost	Total cost of reservation	Yes	Yes	Textview	String	N/A
7	PIN	PIN code of reservation	Yes	Yes	Textview	String	N/A

Table 212: History transaction list view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
1	Back	Back to previous activity	N/A	Transfer to previous activity

Table 213: History transaction list view screen button/hyperlinks

5.2.2.6. History transaction detail view screen

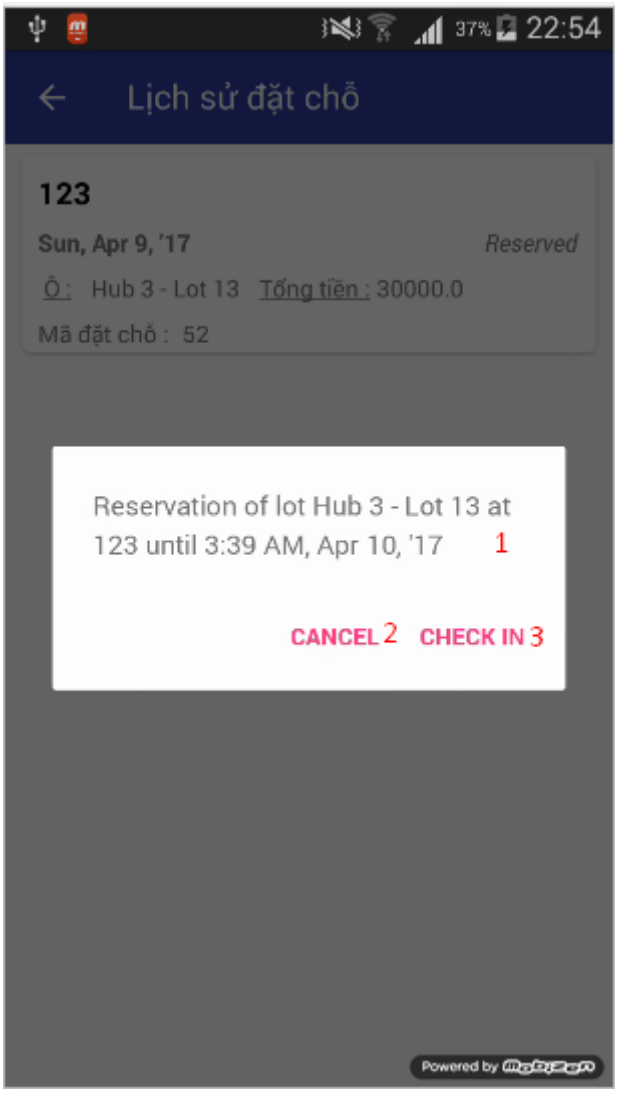


Figure 101: History transaction detail view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Detail	Detail information of transaction	Yes	Yes	Textview	String	N/A

Table 214: History transaction detail view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
2	Cancel	Cancel transaction	N/A	Perform cancel transaction process
3	Check –in	Check-in car park	N/A	Perform check-in transaction process

Table 215: History transaction detail view screen button/hyperlinks

5.2.2.7. Reservation process view screen

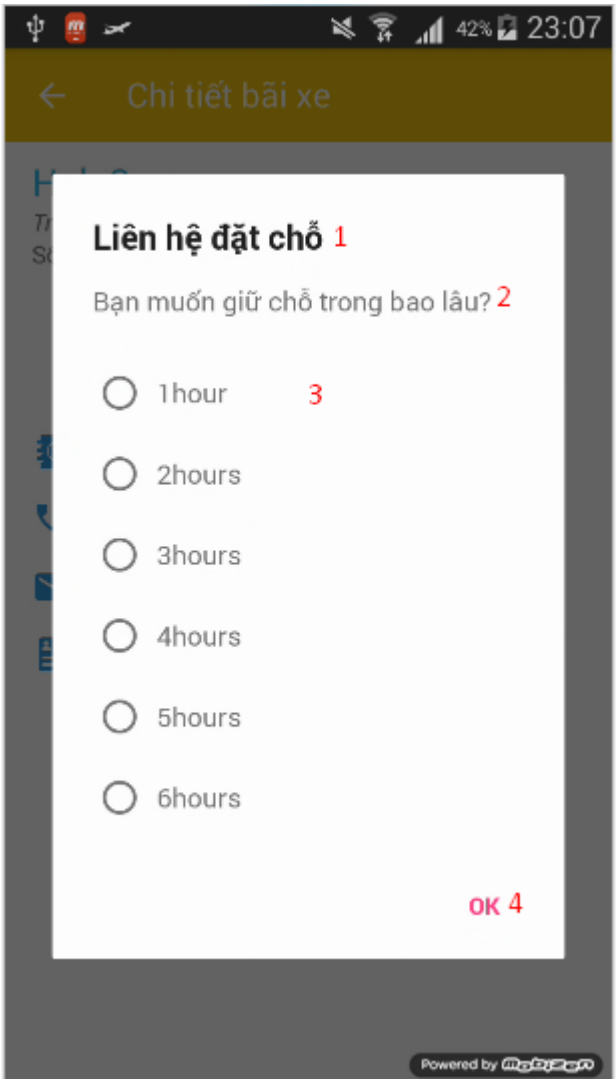


Figure 102: Reservation process view screen 1

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Title	Dialog title	Yes	Yes	Textview	String	N/A
2	Content	Content message of dialog	Yes	Yes	Textview	String	N/A
3	Choice list	Choice list of dialog	N/A	Yes	List	String	N/A

Table 216: Reservation process view screen 1 fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
4	OK	Finish step 1 of reservation	N/A	Show reservation process dialog 2

Table 217: Reservation process view screen 1 button/hyperlinks

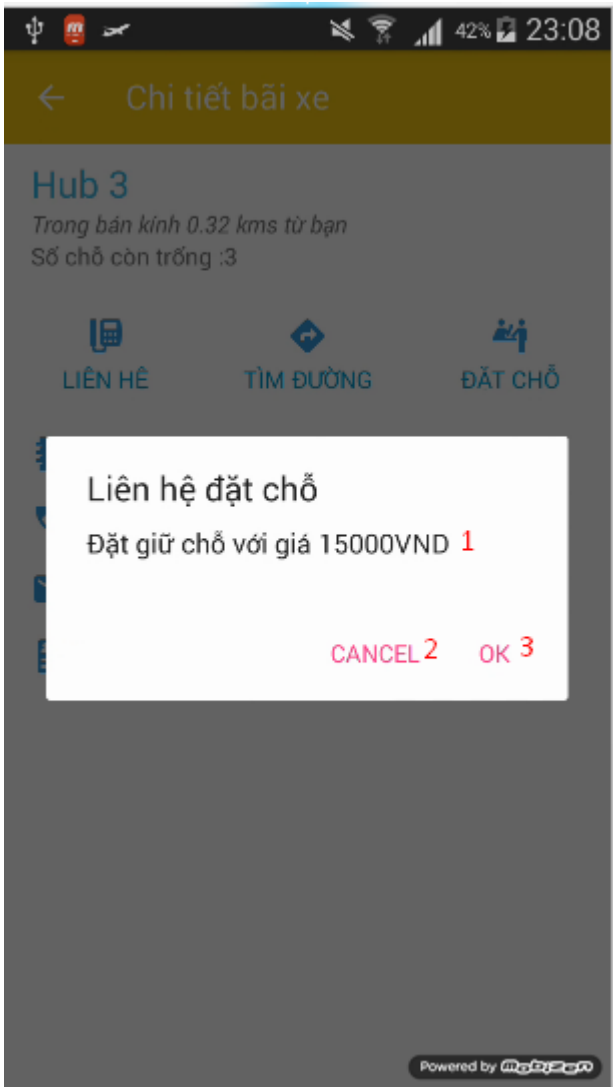


Figure 103: Reservation process view screen 2

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Cost	Message of the total cost of reservation	Yes	Yes	Textview	String	N/A

Table 218: Reservation process view screen 2 fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
2	Cancel	Cancel the reservation	N/A	Close the dialog
3	OK	Confirm the reservation	N/A	Send the request to servers.

Table 219: Reservation process view screen 2 button/hyperlinks

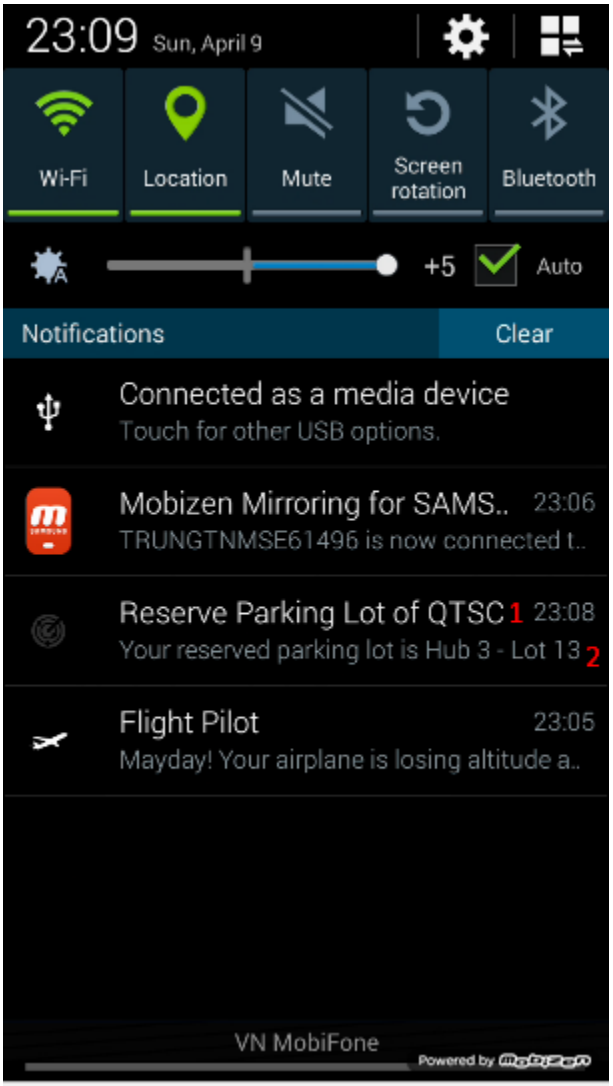


Figure 104: Reservation process notification

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Title	Title of notification	Yes	Yes	Texview	String	N/A

2	Content	Content of notification	Yes	Yes	Textview	String	N/A

Table 220: Reservation process notification fields

Button/Hyperlinks

N/A

5.2.3. Manager Interface

5.2.3.1. Car park list view screen

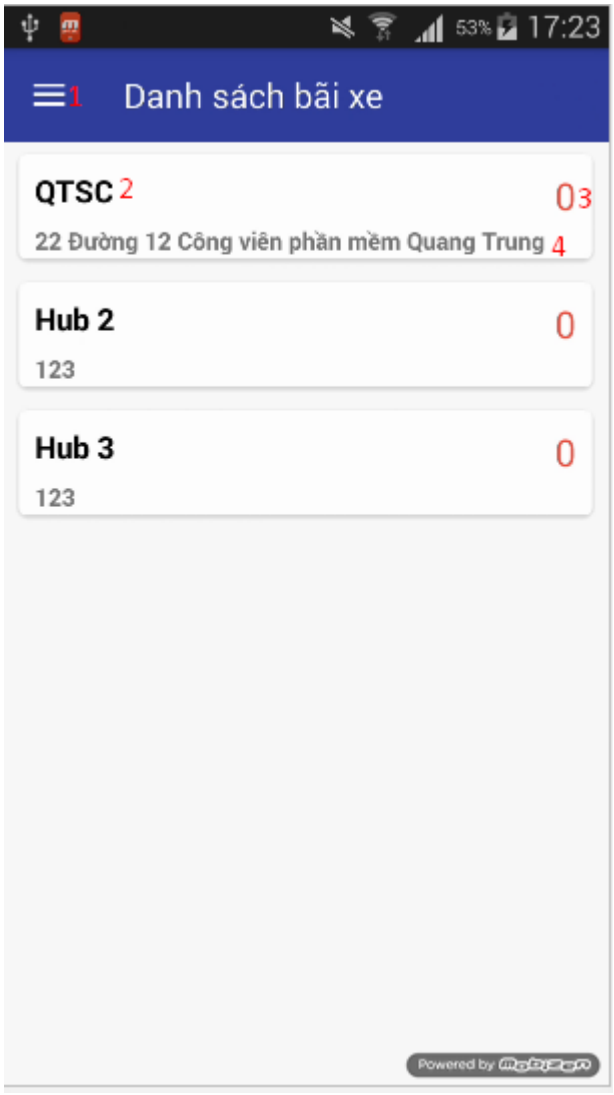


Figure 105: Car park list view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
2	Name	Name of car park	Yes	Yes	Textview	String	N/A
3	Available lot	Current available lot of car park	Yes	Yes	Textview	String	N/A
4	Address	Address of car park	Yes	Yes	Textview	String	N/A

Table 221: Car park list view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
1	Drawer	Show navigation view	N/A	Show navigation

Table 222: Login screen button/hyperlinks

5.2.3.2. Car park detail view screen

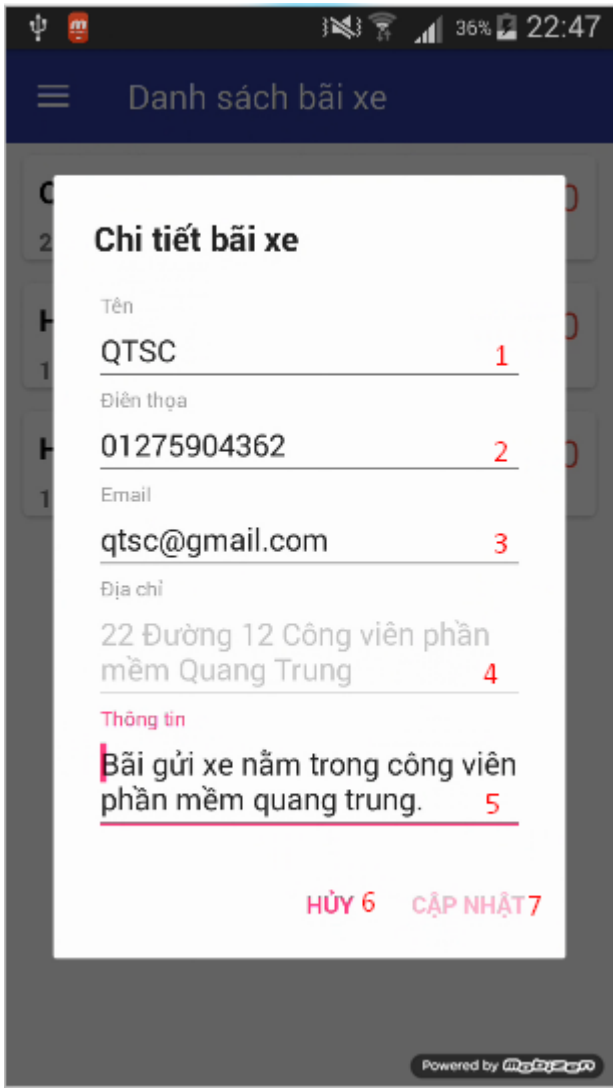


Figure 106: Car park detail view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Name	Name of car park	N/A	Yes	Edittext	String	N/A
2	Phone	Phone of car park	N/A	Yes	Edittext	String	N/A
3	Email	Email of car park	N/A	Yes	Edittext	String	N/A
4	Address	Address of car park	Yes	Yes	Edittext	String	N/A
5	Description	Description of car park	N/A	Yes	Edittext	String	N/A

Table 223: Car park detail view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
6	Cancel	Cancel edit car park	N/A	Close dialog
7	Update	Update car park	N/A	Send request update car park information

Table 224: Car park detail view screen button/hyperlinks

5.2.3.3. Area list view screen

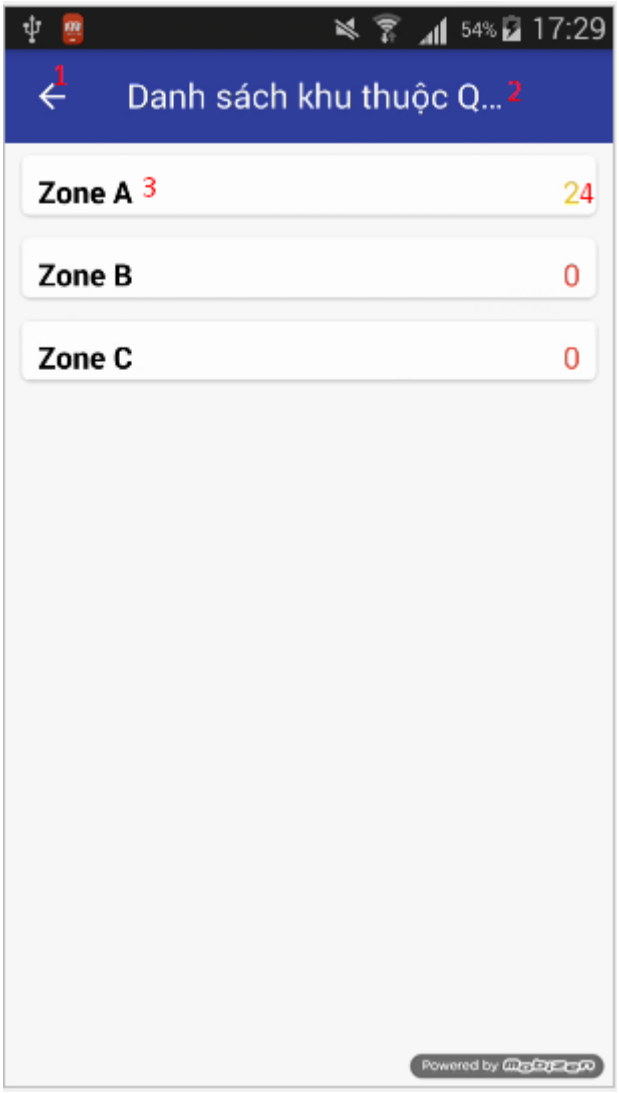


Figure 107: Area list view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
2	Title	Car park name	Yes	Yes	Textview	String	N/A

3	Name	Name of area	Yes	Yes	Textview	String	N/A
4	Available lot	Current available lot of area	Yes	Yes	Textview	String	N/A

Table 225: Area list view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
1	Back	Back to previous activity	N/A	Transfer back to previous activity

Table 226: Area list view screen button/hyperlinks

5.2.3.4. Area detail view screen

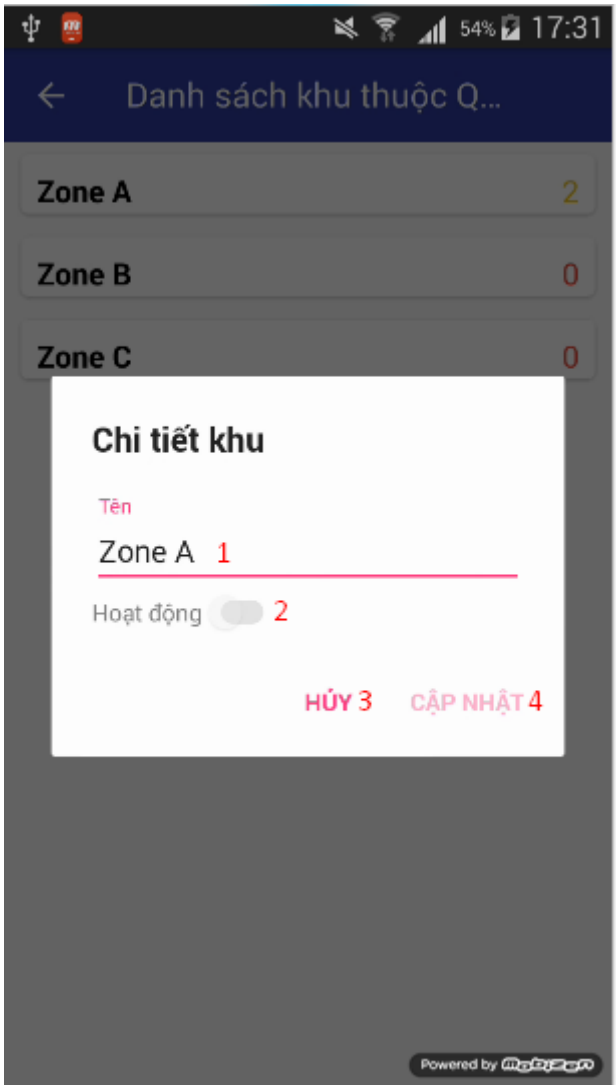


Figure 108: Area detail view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Name	Name of area	N/A	Yes	Edittext	String	N/A
2	Active	Active switch of area	N/A	Yes	Switch	Bool	N/A

Table 227: Area detail view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
3	Cancel	Cancel of edit area	N/A	Close dialog
4	Update	Update area information	N/A	Send request to update area

Table 228: Area detail view screen button/hyperlinks

5.2.3.5. Parking lot list view screen

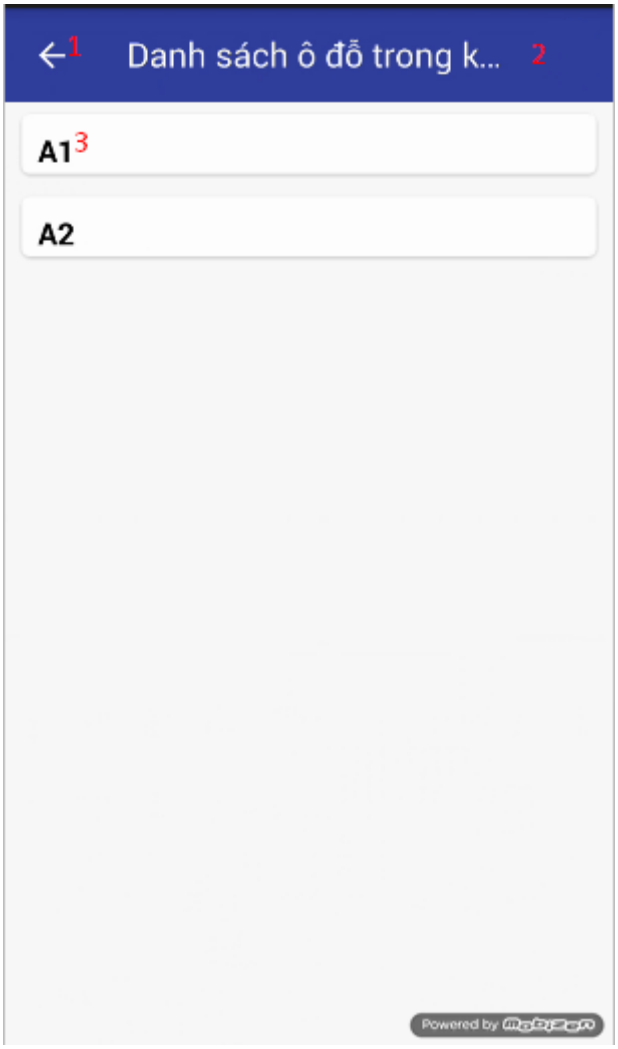


Figure 109: Parking lot list view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
2	Title	Area name	Yes	Yes	Textview	String	N/A
3	Name	Name of parking lot	Yes	Yes	Textview	String	N/A

Table 229: Parking lot list view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
1	Back	Back to previous activity	N/A	Transfer back to previous activity

Table 230: Parking lot list view screen button/hyperlinks

5.2.3.6. Parking lot detail view screen

Figure 110: Parking lot detail view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Name	Name of parking lot	N/A	Yes	Edittext	String	N/A
2	Active	Active switch of parking lot	N/A	Yes	Switch	Bool	N/A

Table 231: Parking lot detail view screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
3	Cancel	Cancel of edit parking lot	N/A	Close dialog
4	Update	Update parking lot information	N/A	Send request to update parking lot

Table 232: Parking lot detail view screen button/hyperlinks

5.2.3.7. Check code view screen

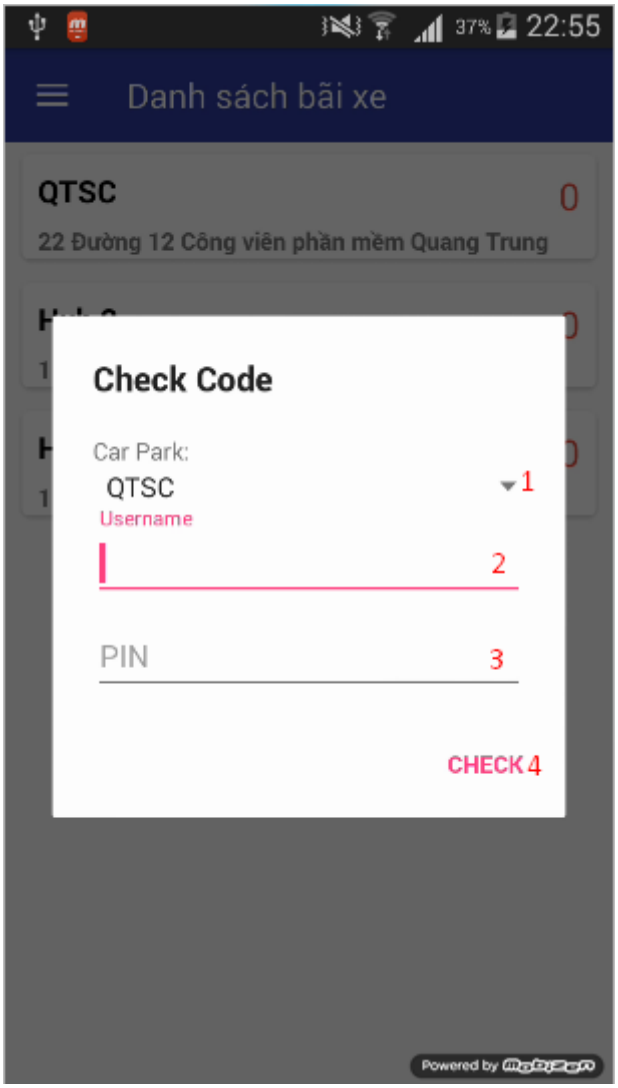


Figure 111: Check code view screen

Fields

No	Field Name	Description	Read Only	Mandatory	Control Type	Data Type	Length
1	Car park list	List of car park	N/A	Yes	Spinner	String	N/A
2	Username	Fill in username	N/A	Yes	Edittext	String	N/A
3	PIN	Fill in PIN code	N/A	Yes	Edittext	String	N/A

Table 233: Check code screen fields

Button/Hyperlinks

No	Function	Description	Validation	Outcome
4	Check	Check PIN code with user name and car park	N/A	Send request to check and perform check-in if correct username and pin code for car park

Table 234: Check code screen button/hyperlinks

6. Database Design

6.1. Logical Diagram

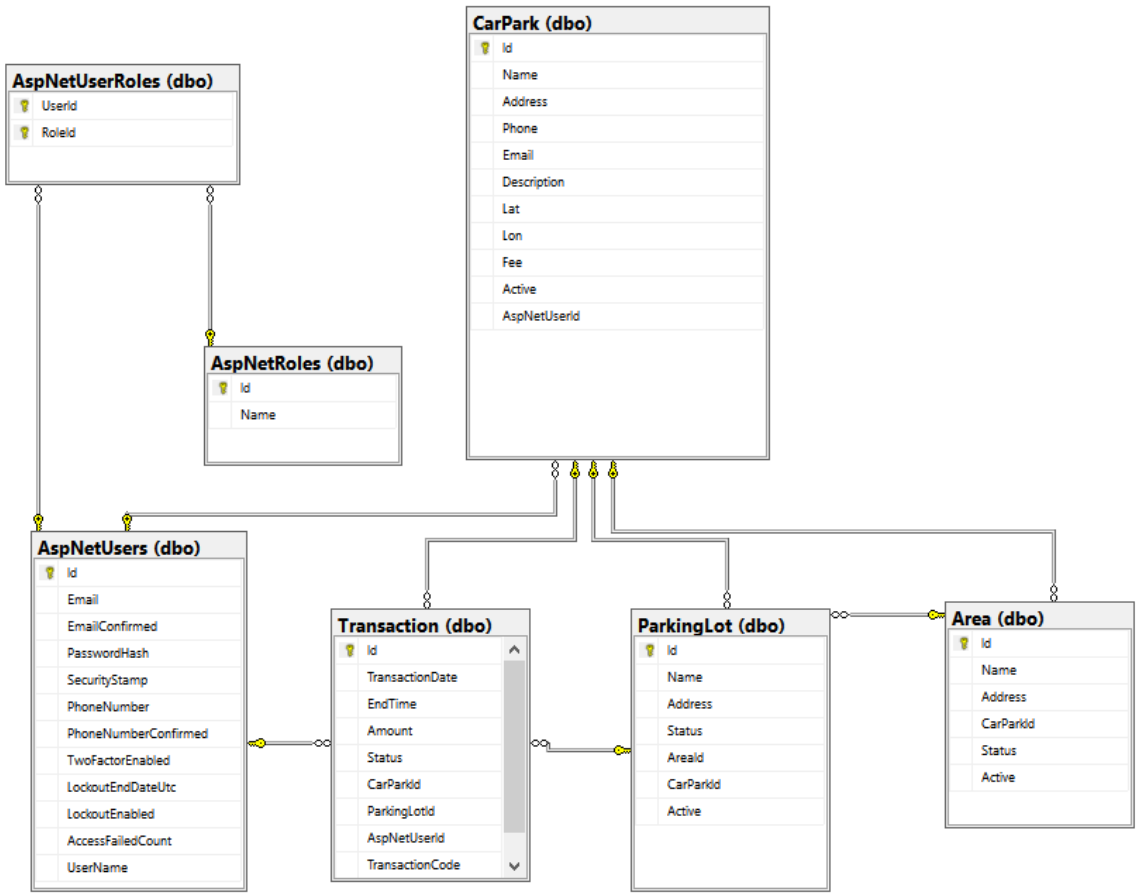


Figure 112: Database logical diagram

6.2. Data Dictionary

Table Name	Column Name	Is Nullable	Data Type
Area	Id	NO	int
	Name	NO	nvarchar
	Address	YES	int
	CarParkId	NO	int
	Status	NO	int
	Active	NO	bit
ParkingLot	Id	NO	int
	Name	NO	nvarchar
	Address	NO	int
	Status	NO	int

	AreaId	YES	int
	CarParkId	NO	int
	Active	NO	bit
Transaction	Id	NO	int
	TransactionDate	NO	datetime
	EndTime	YES	datetime
	Amount	NO	money
	Status	NO	int
	CarParkId	NO	int
	ParkingLotId	NO	int
	AspNetUserId	NO	nvarchar
	TransactionCode	YES	nvarchar
CarPark	Id	NO	int
	Name	NO	nvarchar
	Address	NO	nvarchar
	Phone	YES	nvarchar
	Email	YES	nvarchar
	Description	YES	nvarchar
	Lat	NO	nvarchar
	Lon	NO	nvarchar
	Fee	YES	money
	Active	NO	bit
	AspNetUserId	YES	nvarchar
AspNetRoles	Id	NO	nvarchar
	Name	NO	nvarchar
AspNetUserRoles	UserId	NO	nvarchar
	RoleId	NO	nvarchar
AspNetUsers	Id	NO	nvarchar
	Email	YES	nvarchar
	EmailConfirmed	NO	bit
	PasswordHash	YES	nvarchar
	SecurityStamp	YES	nvarchar
	PhoneNumber	YES	nvarchar
	PhoneNumberConfirmed	NO	bit
	TwoFactorEnabled	NO	bit
	LockoutEndDateUtc	YES	datetime
	LockoutEnabled	NO	bit
	AccessFailedCount	NO	int
	UserName	NO	nvarchar

Table 235: Database data dictionary

7. Algorithms

7.1. GetDistance formula

To calculate the long distance with more accuracy, we must calculate the distance in the circle by using Haversine formula

$$hav\left(\frac{d}{r}\right) = hav(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\, hav(\lambda_2 - \lambda_1)$$

hav is the haversine function

$$hav(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

d is the distance between the two points

r is the radius of the sphere,

φ_1, φ_2: latitude of point 1 and latitude of point 2, in radians

λ_1, λ_2: longitude of point 1 and longitude of point 2, in radians

By apply the inverse haversine or by using the arcsine, we can calculate d:

$$d = r.hav^{-1}(h) = 2r.\arcsin\left(\sqrt{h}\right)$$

$$d = 2r.\arcsin(\sqrt{hav(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\, hav(\lambda_2 - \lambda_1)}$$

$$d = 2r.\arcsin(\sqrt{sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\, sin^2(\lambda_2 - \lambda_1)}$$

7.2. CRC Error Detection

7.2.1. Definition

The aim of an error detection technique is to enable the receiver of a message transmitted through a noisy (error-introducing) channel to determine whether the message has been corrupted. To do this, the transmitter constructs a value (called a checksum) that is a function of the message, and appends it to the message. The receiver can then use the same function to calculate the checksum of the received message and compare it with the appended checksum to see if the message was correctly received.

7.2.2. Define Problem

Because we do not require a license to operate radio equipment in the 2.4GHz band, as a result, other transmitter can broadcast on the same frequency that our devices use.

7.2.3. Solution theory

Two important aspects required to form a strong checksum function:

- **WIDTH:** a register width wide enough to provide a low a-priori probability of failure (e.g. 32-bits gives a 1/2^32 chance of failure).
- **CHAOS:** a formula that gives each input byte the potential to change any number of bits in the register.

For that reason, the CRC schemes use division with some changes so that if more bytes were added to the message, the checksum value could change radically again very quickly. All the arithmetic performed during CRC calculation is performed in binary with no carries, we will call it CRC arithmetic.

Adding two numbers in CRC arithmetic is the same as adding numbers in ordinary binary arithmetic except there is no carry. This means that each pair of corresponding bits determine the corresponding output bit without reference to any other bit positions. For example:

	1	0	0	1	1	0	1	1
+	1	1	0	0	1	0	1	0
	0	1	0	1	0	0	0	1

Figure 113: CRC arithmetic addition

Subtraction is identical:

	1	0	0	1	1	0	1	1
-	1	1	0	0	1	0	1	0
	0	1	0	1	0	0	0	1

Figure 114: CRC arithmetic subtraction

In fact, both addition and subtraction in CRC arithmetic is equivalent to the XOR operation, and the XOR operation is its own inverse. This effectively reduces the operations of the first level of power (addition, subtraction) to a single operation that is its own inverse. This is a very convenient property of the arithmetic.

Having defined addition, we can move to multiplication and division. Multiplication is absolutely straightforward, being the sum of the first number, shifted in accordance with the second number. (note: the sum uses CRC addition)

			1	1	0	1	
		x	1	0	1	1	
			1	1	0	1	
		1	1	0	1		
	0	0	0	0			
1	1	0	1				
1	1	1	1	1	1	1	

Figure 115: CRC arithmetic multiplication

Division is a little messier as we need to know when "a number goes into another number". To do this, we invoke the weak definition of magnitude defined earlier: that X is greater

than or equal to Y if the position of the highest 1 bit of X is the same or greater than the position of the highest 1 bit of Y.

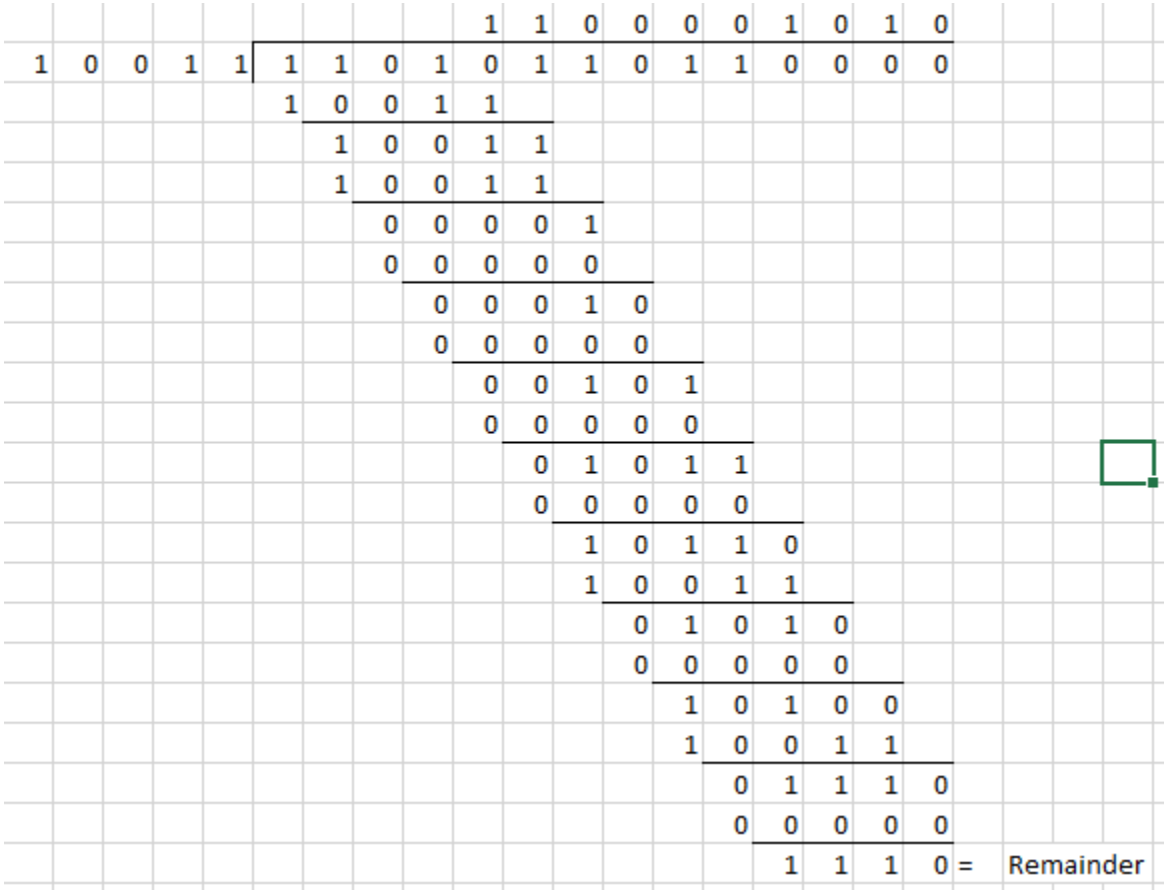


Figure 116: CRC arithmetic division

Having defined CRC arithmetic, we can now frame a CRC calculation as simply a division, because that's all it is. To perform a CRC calculation, we need to choose a divisor. In math marketing speak the divisor is called the "generator polynomial" or simply the "polynomial", and is a key parameter of any CRC algorithm. As a compromise, we will refer to the CRC polynomial as the "poly".

The width (position of the highest 1 bit) of the poly is very important as it dominates the whole calculation. Typically, widths of 16 or 32 are chosen so as to simplify implementation on modern computers. The width of a poly is the actual bit position of the highest bit. For example, the width of 10011 is 4, not 5. For the purposes of example, we will choose a poly of 10011 (of width W of 4).

Having chosen a poly, we can proceed with the calculation. This is simply a division (in CRC arithmetic) of the message by the poly. The only trick is that W zero bits are appended to the message before the CRC is calculated. Thus we have:

- Original message: 1101011011
- Poly: 10011

- Message after appending W zeros: 11010110110000

This is the same division as above figure, so we got the remainder, which is the calculated checksum is 1110. Usually, the checksum is then appended to the message and the result transmitted. In this case the transmission would be: 11010110111110. This ends the calculation.

A summary of the operation of the class of CRC algorithms:

- Choose a width W, and a poly G (of width W).
- Append W zero bits to the message. Call this M'.
- Divide M' by G using CRC arithmetic. The remainder is the checksum.

7.2.4. Implementation

To implement a CRC algorithm all we have to do is implement CRC division. There are two reasons why we cannot simply use the divide instruction of whatever machine we are on. The first is that we have to do the divide in CRC arithmetic. The second is that the dividend might be ten megabytes long, and todays processors do not have registers that big.

So to implement CRC division, we have to feed the message through a division register. At this point, we have to be absolutely precise about the message data. In all the following examples the message will be considered to be a stream of bytes (each of 8 bits) with bit 7 of each byte being considered to be the most significant bit (MSB). The bit stream formed from these bytes will be the bit stream with the MSB (bit 7) of the first byte first, going down to bit 0 of the first byte, and then the MSB of the second byte and so on.

For the purposes of example, consider a poly with W=4 and the poly=10111. Then, the perform the division, we need to use a 4-bit register:

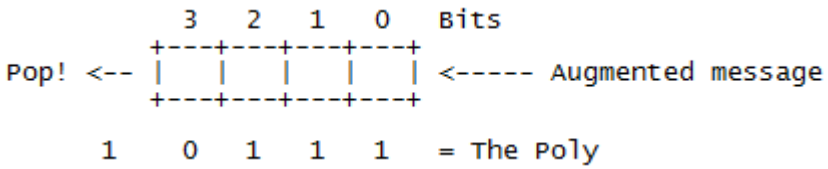


Figure 117: CRC implementation register

(Augmented message is the message followed by W zero bits)

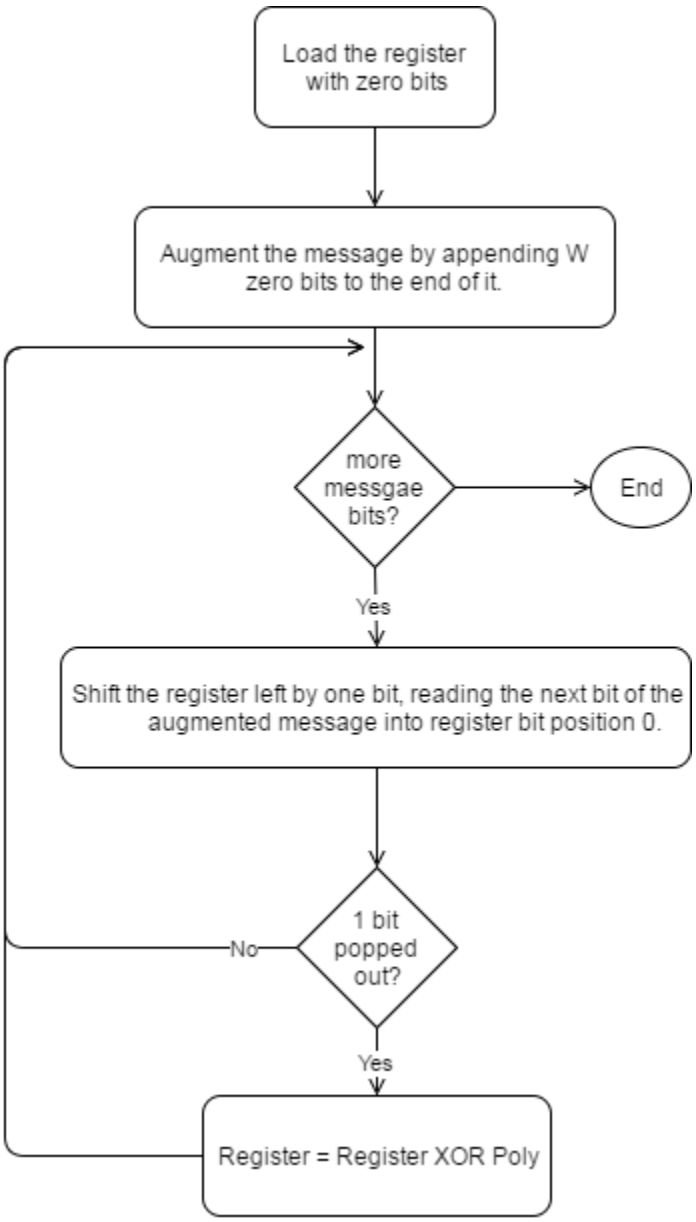


Figure 118: CRC implementation simple flow

7.2.5. Table-driven implementation

The implementation above is a good starting point because it corresponds directly to the theory presented so far, and because it is so simple. However, because it operates at the bit level, it is rather awkward to code (even in C), and inefficient to execute (it has to loop once for each bit). To speed it up, we need to find a way to enable the algorithm to process the message in units larger than one bit.

For the purposes of discussion, let us switch from a 4-bit poly to a 32-bit one. Our register looks much the same, except the boxes represent bytes instead of bits, and the Poly is 33 bits (one implicit 1 bit at the top and 32 "active" bits) (W=32).

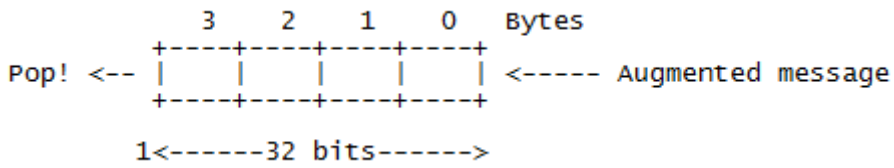


Figure 119: CRC 32-bits register

Consider for a moment that we use the top 8 bits of the register to calculate the value of the top bit of the register during the next 8 iterations. Suppose that we drive the next 8 iterations using the calculated values (which we could perhaps store in a single byte register and shift out to pick off each bit). Then we note three things:

- The top byte of the register now doesn't matter. No matter how many times and at what offset the poly is XORed to the top 8 bits, they will all be shifted out the right hand side during the next 8 iterations anyway.
- The remaining bits will be shifted left one position and the rightmost byte of the register will be shifted in the next byte.
- While all this is going on, the register will be subjected to a series of XOR's in accordance with the bits of the pre-calculated control byte.

Now consider the effect of XORing in a constant value at various offsets to a register. For example:

```
0100010  Register
...0110  XOR this
..0110.  XOR this
0110...  XOR this
-------
0011000
-------
```

The point of this is that you can XOR constant values into a register to your heart's delight, and in the end, there will exist a value which when XORed in with the original register will have the same effect as all the other XORs.

Putting all the pieces together we have an algorithm that goes like this:

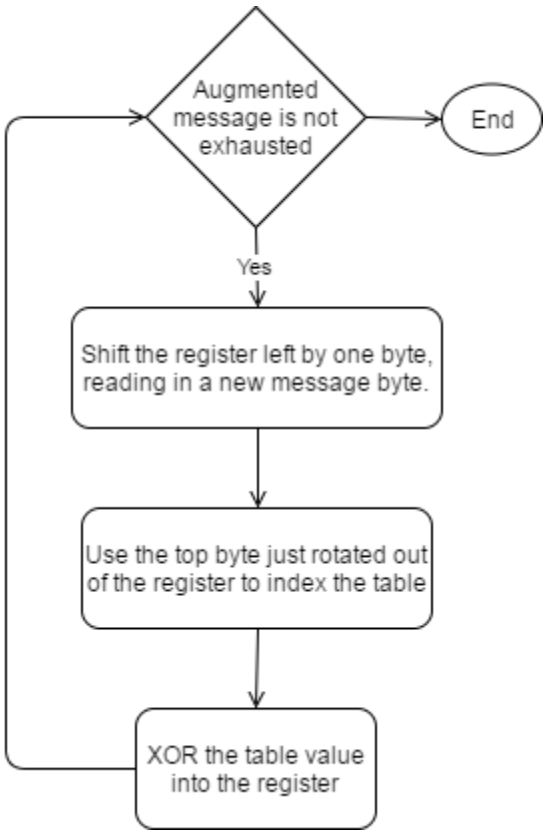


Figure 120: CRC implementation table-driven flow

For the actual implementation in project, we use 3 bytes in payload package as checksum, so we need to implement CRC24. Here is the table we use for CRC24 table-driven implementation:

```
0x00000000, 0x00864cfb, 0x008ad50d, 0x000c99f6, 0x0093e6e1, 0x0015aa1a, 0x001933ec,
0x009f7f17, 0x00a18139, 0x0027cdc2, 0x002b5434, 0x00ad18cf, 0x003267d8, 0x00b42b23,
0x00b8b2d5, 0x003efe2e, 0x00c54e89, 0x00430272, 0x004f9b84, 0x00c9d77f, 0x0056a868,
0x00d0e493, 0x00dc7d65, 0x005a319e, 0x0064cfb0, 0x00e2834b, 0x00ee1abd, 0x00685646,
0x00f72951, 0x007165aa, 0x007dfc5c, 0x00fbb0a7, 0x000cd1e9, 0x008a9d12, 0x008604e4,
0x0000481f, 0x009f3708, 0x00197bf3, 0x0015e205, 0x0093aefe, 0x00ad50d0, 0x002b1c2b,
0x002785dd, 0x00a1c926, 0x003eb631, 0x00b8faca, 0x00b4633c, 0x00322fc7, 0x00c99f60,
0x004fd39b, 0x00434a6d, 0x00c50696, 0x005a7981, 0x00dc357a, 0x00d0ac8c, 0x0056e077,
0x00681e59, 0x00ee52a2, 0x00e2cb54, 0x006487af, 0x00fbf8b8, 0x007db443, 0x00712db5,
0x00f7614e, 0x0019a3d2, 0x009fef29, 0x009376df, 0x00153a24, 0x008a4533, 0x000c09c8,
0x0000903e, 0x0086dcc5, 0x00b822eb, 0x003e6e10, 0x0032f7e6, 0x00b4bb1d, 0x002bc40a,
0x00ad88f1, 0x00a11107, 0x00275dfc, 0x00dced5b, 0x005aa1a0, 0x00563856, 0x00d074ad,
0x004f0bba, 0x00c94741, 0x00c5deb7, 0x0043924c, 0x007d6c62, 0x00fb2099, 0x00f7b96f,
0x0071f594, 0x00ee8a83, 0x0068c678, 0x00645f8e, 0x00e21375, 0x0015723b, 0x00933ec0,
0x009fa736, 0x0019ebcd, 0x008694da, 0x0000d821, 0x000c41d7, 0x008a0d2c, 0x00b4f302,
0x0032bff9, 0x003e260f, 0x00b86af4, 0x002715e3, 0x00a15918, 0x00adc0ee, 0x002b8c15,
0x00d03cb2, 0x00567049, 0x005ae9bf, 0x00dca544, 0x0043da53, 0x00c596a8, 0x00c90f5e,
0x004f43a5, 0x0071bd8b, 0x00f7f170, 0x00fb6886, 0x007d247d, 0x00e25b6a, 0x00641791,
0x00688e67, 0x00eec29c, 0x003347a4, 0x00b50b5f, 0x00b992a9, 0x003fde52, 0x00a0a145,
0x0026edbe, 0x002a7448, 0x00ac38b3, 0x0092c69d, 0x00148a66, 0x00181390, 0x009e5f6b,
0x0001207c, 0x00876c87, 0x008bf571, 0x000db98a, 0x00f6092d, 0x007045d6, 0x007cdc20,
0x00fa90db, 0x0065efcc, 0x00e3a337, 0x00ef3ac1, 0x0069763a, 0x00578814, 0x00d1c4ef,
0x00dd5d19, 0x005b11e2, 0x00c46ef5, 0x0042220e, 0x004ebbf8, 0x00c8f703, 0x003f964d,
0x00b9dab6, 0x00b54340, 0x00330fbb, 0x00ac70ac, 0x002a3c57, 0x0026a5a1, 0x00a0e95a,
0x009e1774, 0x00185b8f, 0x0014c279, 0x00928e82, 0x000df195, 0x008bbd6e, 0x00872498,
0x00016863, 0x00fad8c4, 0x007c943f, 0x00700dc9, 0x00f64132, 0x00693e25, 0x00ef72de,
0x00e3eb28, 0x0065a7d3, 0x005b59fd, 0x00dd1506, 0x00d18cf0, 0x0057c00b, 0x00c8bf1c,
0x004ef3e7, 0x00426a11, 0x00c426ea, 0x002ae476, 0x00aca88d, 0x00a0317b, 0x00267d80,
0x00b90297, 0x003f4e6c, 0x0033d79a, 0x00b59b61, 0x008b654f, 0x000d29b4, 0x0001b042,
0x0087fcb9, 0x001883ae, 0x009ecf55, 0x009256a3, 0x00141a58, 0x00efaaff, 0x0069e604,
0x00657ff2, 0x00e33309, 0x007c4c1e, 0x00fa00e5, 0x00f69913, 0x0070d5e8, 0x004e2bc6,
0x00c8673d, 0x00c4fecb, 0x0042b230, 0x00ddcd27, 0x005b81dc, 0x0057182a, 0x00d154d1,
0x0026359f, 0x00a07964, 0x00ace092, 0x002aac69, 0x00b5d37e, 0x00339f85, 0x003f0673,
0x00b94a88, 0x0087b4a6, 0x0001f85d, 0x000d61ab, 0x008b2d50, 0x00145247, 0x00921ebc,
0x009e874a, 0x0018cbb1, 0x00e37b16, 0x006537ed, 0x0069ae1b, 0x00efe2e0, 0x00709df7,
0x00f6d10c, 0x00fa48fa, 0x007c0401, 0x0042fa2f, 0x00c4b6d4, 0x00c82f22, 0x004e63d9,
0x00d11cce, 0x00575035, 0x005bc9c3, 0x00dd8538
```

Figure 121: CRC24 precomputed table

E. System Implementation & Test

1. Introduction

1.1. Overview

This section has all necessary information about test plan, test case and its result, the environment for testing and test pass/fail criteria

1.2. Test Approach

White-box: Developers self-test on code in which function they developed

In this section, Black-box testing will be used to test whether the whole system meets the following objectives

2. Database Relationship Diagram

2.1. Physical Diagram

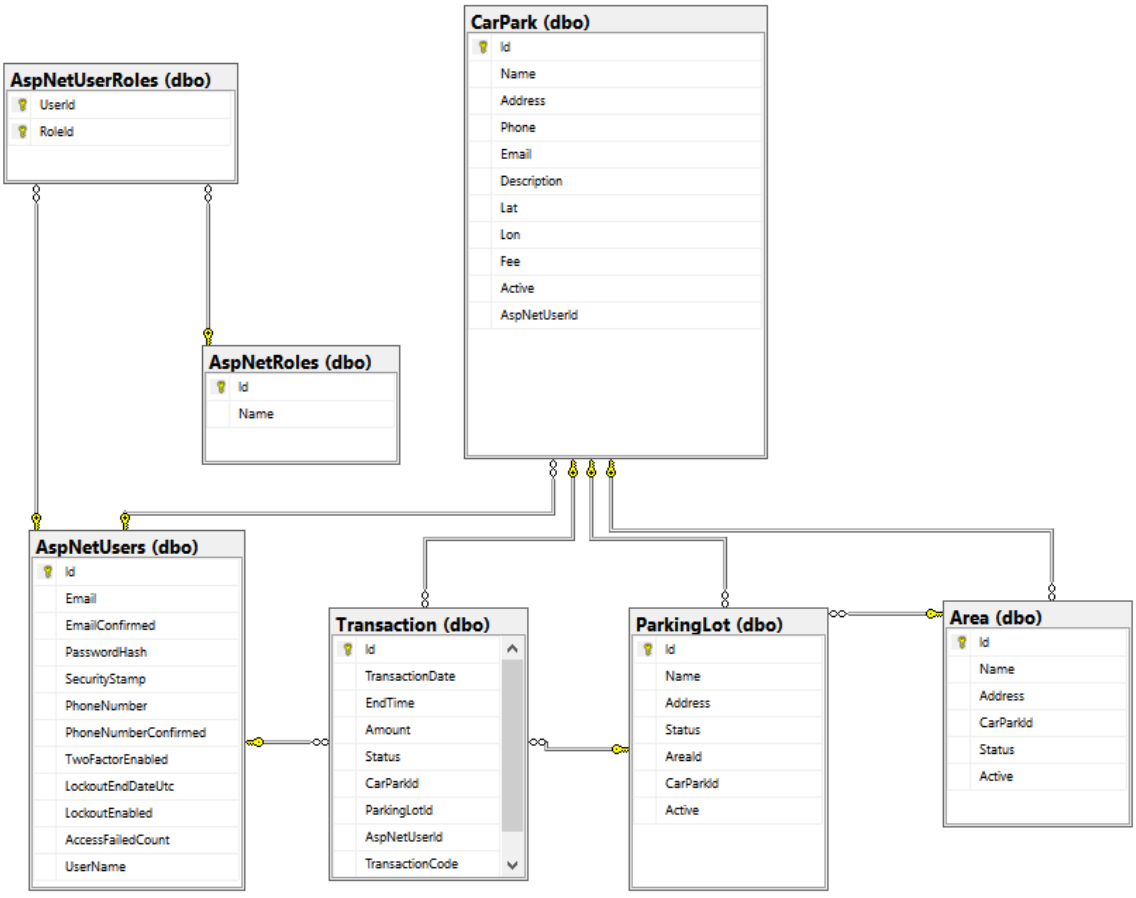


Figure 122: Database physical diagram

2.2. Data Dictionary

Table Name	Description
Area	Contain the area information of the car park
ParkingLot	Contain the information of each lot in the car park
Transaction	Contain the transaction information
CarPark	Contain the information of each car park
AspNetRoles	Contain the user's role name
AspNetUserRoles	The mapping between 2 tables: AspNetRoles and AspNetUsers
AspNetUsers	Contain the user's information

Table 236: Data dictionary

Table Name	Column Name	Description	Is Nullable	Data Type
Area	Id	Unique Id for each area	NO	int
	Name	Name of area	NO	nvarchar
	Address	Bit address of a area	YES	int
	CarParkId	Id of car park which hold area	NO	int
	Status	The number indicate the status of area	NO	int
	Active	Value show that the area is deleted or not	NO	bit
ParkingLot	Id	Unique Id for each parking lot	NO	int
	Name	Name of parking lot	NO	nvarchar
	Address	Bit address of a parking lot	NO	int

	Status	The number indicate the status of parking lot	NO	int
	AreaId	Value show that the parking lot is deleted or not	YES	int
	CarParkId	Id of area which hold parking lot	NO	int
	Active	Id of car park which hold parking lot	NO	bit
Transaction	Id	Unique Id for each transaction	NO	int
	TransactionDate	The time customer book the parking lot	NO	datetime
	EndTime	End of the booking time	YES	datetime
	Amount	The total fee of transaction	NO	money
	Status	The number indicate status of the transaction	NO	int
	CarParkId	The code for the user to enter in the car park to unlock the booked parking lot	NO	int
	ParkingLotId	Id of the lot user reserved	NO	int
	AspNetUserId	Id of the user	NO	nvarchar

		who reserved		
	TransactionCode	The code for the user to enter in the car park to unlock the booked parking lot	YES	nvarchar
CarPark	Id	Unique Id for each car park	NO	int
	Name	Name of car park	NO	nvarchar
	Address	Address of carpark	NO	nvarchar
	Phone	Phone number of car park	YES	nvarchar
	Email	Email of car park's owner	YES	nvarchar
	Description	Information of car park	YES	nvarchar
	Lat	Latitude of a car park	NO	nvarchar
	Lon	Longitude of a car park	NO	nvarchar
	Fee	Amount of money each hour	YES	money
	Active	Value show that the car park is deleted or not	NO	bit
	AspNetUserId	Id of the owner	YES	nvarchar
AspNetRoles	Id	Unique Id of each role	NO	nvarchar
	Name	Role name	NO	nvarchar
AspNetUserRoles	UserId	Id of the AspNetUsers	NO	nvarchar

	RoleId	Id of the AspNetRoles	NO	nvarchar
AspNetUsers	Id	Unique Id for each user	NO	nvarchar
	Email	Email of the user	YES	nvarchar
	EmailConfirmed	Check if the user confirm the email	NO	bit
	PasswordHash	Hash of the user password, don't save exact password	YES	nvarchar
	SecurityStamp		YES	nvarchar
	PhoneNumber	The number of the user	YES	nvarchar
	PhoneNumberConfirmed	Check if the number is confirmed	NO	bit
	TwoFactorEnabled		NO	bit
	LockoutEndDateUtc		YES	datetime
	LockoutEnabled		NO	bit
	AccessFailedCount	The number of time access failed	NO	int
	UserName	Username of the user	NO	nvarchar

Table 237: Data dictionary detail

3. Performance Measures

The most important function in our project is the working of RF network, specifically the polling process of parking lots. The RF module we use is a 2.4GHz, which may contain lots of radio-frequency interference, so we perform a performance test with following conditions to check the reliability of the system:

- The location is in a personal home, so there is more interference than a real car park.
- We set the Hub to continuously polling 6 lot nodes place near each other. After the Hub polls data from all 6 lot nodes, it will count as 1 round.
- For simplicity's sake and easier to check logging data, we set the condition for the Hub to send a package and wait ACK package in 100ms. After 100ms, the Hub will resend the package. After 5 times resend and the Hub still not received the ACK package, the polling for that node will count as failed and move to the next node, and that round will also count as failed.

We let the system run continuously for around 16h (~11h PM to 3h PM the next day) and get the following results:

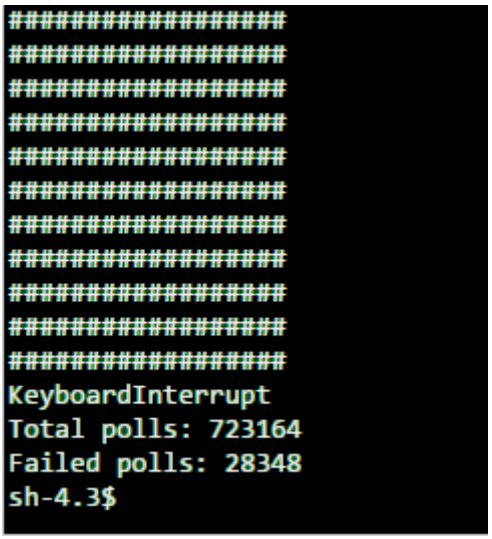


Figure 123: Performance test result

The system performed 723 164 rounds of polling and got 28 348 rounds that got at least 1 lot node didn't response after 500ms. After the test, we get that only **96.08%** of the package send from Hub is successfully received and executed. The test also shows that if the polling of lot in current round is failed, the next round will come quickly so the information Hub owns is reliable. In the current setting, with a large amount of lot nodes, when the system meets radio-frequency interference it may got a little lag.

4. Test Plan

The purpose of the test is to verify the functionality of the system.

Functions need to be tested. Functions to ensure technical requirements and system requirements of the user. Error will not happen after the trial

The next content will describe which function will be tested, which will not and plan for them.

4.1. Features to be tested

We separate two part for testing: hardware and software.

4.1.1. Hardware

After assembling hardware devices to system, we test all of them to ensure that all hardware devices working well.

Here is a list of hardware devices that we tested.

Functions	Description
Magnetometer Sensor HMC5883L	We test detect metal indoor, outdoor, in real environment at some points of time in day. After many test case, we will get the estimate value of the sensor
NRF24L01	RF Communicate: We tested packets to transfer and receive. We define some variables, which have different sizes, to put in packet for transfer and receive via RF from sub to main and otherwise. If NRF24L01 could transfer and receive the packet, test case is “Passes”, else if NRF24L01 could not transfer or receive, test case is “Failed”. We also tested the distance of RF in different topographic in many regions such as obstructions of buildings, walls or metal. Besides that, we also tested distance on the straight line, how far they could work. The test case is “Passed” if sub and main still send and receive signals, otherwise is “Failed”
Arduino Nano	The controller in sub node, we tested the SPI and I2C connection between Nano and NRF24L01, HMC5883L. If it can receive and transfer data, the test case is “Passed”, otherwise is “Failed”.
Raspberry Pi 3	We tested ability of SPI connection between Raspberry Pi and NRF24L01. If it can receive and transfer data, the test case is “Passed”, otherwise is “Failed”
Led Segment	We need to tested the ability to display number on the led from 0 to 9
Servo SG90 Mini	We test the ability to rotate from 0 degree

	to 90 degree of the servo

Table 238: Hardware devices test list

4.1.2. Software

The features of user and web services will be focused and list below:

Functions	Description
API Web Service	Services provide for Raspberry Pi 3. • Get the list of Area • Get the list of Parking Lot • Update status of Parking Lot Service provide for Mobile Application • Register • Login • Get the list of car park • Get car park information • Get the list of area • Update the area • Get the list of parking lot • Update the parking lot
Mobile Application	Black box testing on user interface

Table 239: Software test list

4.2. Features not to be tested:

N/A

4.3. Test environment

Web server: Firefox 52.0.2(32-bit)

OS: Windows 10

Android application: Android 5.0

Hardware: Test on Raspberry Pi 3

4.4. Test Pass/Fail Criteria

For system testing, the criteria are:

- 90% of the test cases must pass
- 100% of test cases about hardware module must pass
- All test cases dealing with critical functionality must pass
- All medium and high severity defects must be fixed
- Test coverage must be at least 90%

5. System Testing Test Case

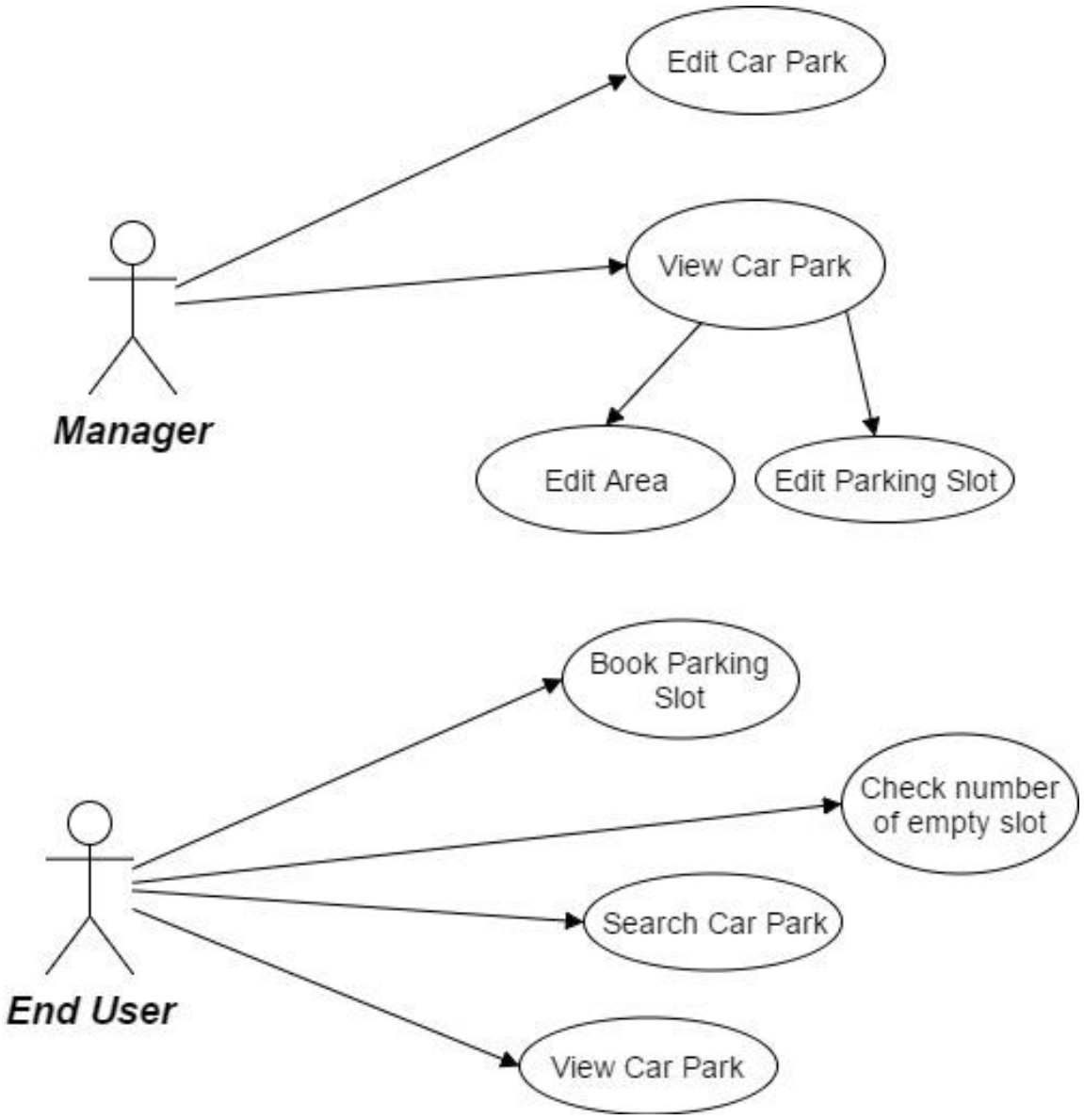


Figure 124: Manager, End User Core F

5.1. Component Testing

Id	Test Case Description	Test Case Procedure	Expected Output	Inter-test Case Dependence	Result	Test Date	Note
C-01	HMC5883L – Detect car park in range	Connect HMC5883L with Nano Connect RGB Led with Nano Add a car on the top of the board	The Led go red	N/A	Passed	03/04/2017	
C-02	HMC5883L – Detect car park in range	Connect HMC5883L with Nano Connect RGB Led with Nano remove a car on the top of the board	The Led go green	N/A	Passed	03/04/2017	
C-03	NRF24L01 + 7-Segment Led	Connect NRF24L01 with Nano Connect 7Segment Led with Nano Send a message with a number to Nano through RF	Display correct number send to Nano by 7Segment Led	N/A	Passed	03/04/2017	

C-04	Arduino Nano	Connect NRF24L01 with Nano Connect HMC5883L with Nano Put a car on top of the board	The status of lot is send to Pi and update on server	N/A	Passed	03/04/2017	
C-05	Raspberry Pi 3	Connect NRF24L01 with Pi 3 Send a message from Pi to Nano through RF	Message receive in Nano	N/A	Passed	03/04/2017	
C-06	Servo SG90 Mini	Connect NRF24L01 with Nano Connect Servo with Nano User reserve a lot on mobile application	The servo rotate 90 degree to block a reserved lot	N/A	Passed	03/04/2017	

Table 240: Component test

5.2. API Web Service Testing

Id	Test Case Description	Test Case Procedure	Expected Output	Inter-test Case Dependence	Result	Test Date	Note
API-01	Get the list of Area	Raspberry connect to Internet Call API through HTTP request	Show the list of Area in JSON format	N/A	Passed	31/03/2017	
API-02	Get the list of Parking Lot	Raspberry connect to Internet Call API through HTTP request	Show the list of Parking Lot in JSON format	N/A	Passed	31/03/2017	

API-03	Update status of Parking Lot	Raspberry connect to Internet
		Call API through HTTP request
			The API return JSON with format
			{
			“success”:true
			}
				N/A	Passed	31/03/2017	

API-04	Register	Mobile send request to API Service through HTTP	If create success, return JSON
			{
			“success”:false
			}
			If create fail, return JSON
			{
			“success”:false
			}
				N/A	Passed	31/03/2017	

API-05	Login	Mobile send request to API Service through HTTP	If login success, return JSON
			{
			"success":false
			}
			If login fail, return JSON
			{
			"success":false
			}
				N/A	Passed	31/03/2017	

API-06	Get the list of car park	Mobile send request to API Service through HTTP	Return the list of Car Park in JSON	N/A	Passed	31/03/2017	
API-07	Get the car park information	Mobile send request to API Service through HTTP	Return the Car Park information in JSON	N/A	Passed	31/03/2017	

API-08	Update the area	Mobile send request to API Service through HTTP	If login success, return JSON
			{
			"success":false
			}
			If login fail, return JSON
			{
			"success":false
			}
				N/A	Passed	31/03/2017	

API-09	Update the parking lot	Mobile send request to API Service through	If login success, return JSON { "success":false	N/A	Passed	31/03/2017	

		HTTP	}
			If login fail, return JSON
			{
			“success”:false
			}				

Table 241: API Web service test

5.3. Mobile Testing

Id	Test Case Description	Test Case Procedure	Expected Output	Inter-test Case Dependence	Result	Test Date	Note
M-01	View Car Park	Open Mobile Application Login with manager account Select a car park	Show the car park information on screen	N/A	Passed	31/03/2017	
M-02	Edit Car Park	Open Mobile Application Login with manager account Select a car park Select Edit Fill in update information and click Finish	Update the information to the server and reload page	N/A	Passed	31/03/2017	
M-03	Edit Area	Open Mobile Application	Update the information to the server and reload page	N/A	Passed	31/03/2017	

		Login with manager account Select a car park Select edit next to an area on the list Fill in update information and click Finish					
M-04	Edit Parking Lot	Open Mobile Application Login with manager account Select a car park Select an area Select edit next to a parking lot on the list Fill in update information and click Finish	Update the information to the server and reload page	N/A	Passed	31/03/2017	
M-05	Book Parking Lot	Open Mobile Application Login with end user account Select a car park	Show the reserved lot information if has Otherwise, show the message error information	N/A	Passed	31/03/2017	

		Select Reserve Fill in Addition Information Click Finish					
M-06	Check number of empty lot	Open Mobile Application Login with end user account Select a car park	The car park information screen show the number of empty lot	N/A	Passed	31/03/2017	
M-07	Search Car park	Open Mobile Application Login with end user account Enter the address on the textbox of the map	The map will transfer to the input address and show nearest car park from the address	N/A	Passed	31/03/2017	
M-08	View Car park	Open Mobile Application Login with end user account Select a car park	Show the car park information	N/A	Passed	31/03/2017	

Table 242: Mobile test

F. User's Manual

1. Installation Guide

1.1. Hardware installation

The installation of lot nodes and sign nodes is complicated and depends on the car park. So we decide that if car park owner need to setup the PGSS for his car park, he need to contact our team and we will setup.

1.2. Hub installation

After install the hardware for the car park, it will have a unique Id. The hub run on Raspbian so our team or car park owner need to go to raspberry pi site and download the newest version of Raspbian for their raspberry, follow this link:

https://www.raspberrypi.org/downloads/raspbian/

Run the raspberry the 1st time, need to update it with the following command:

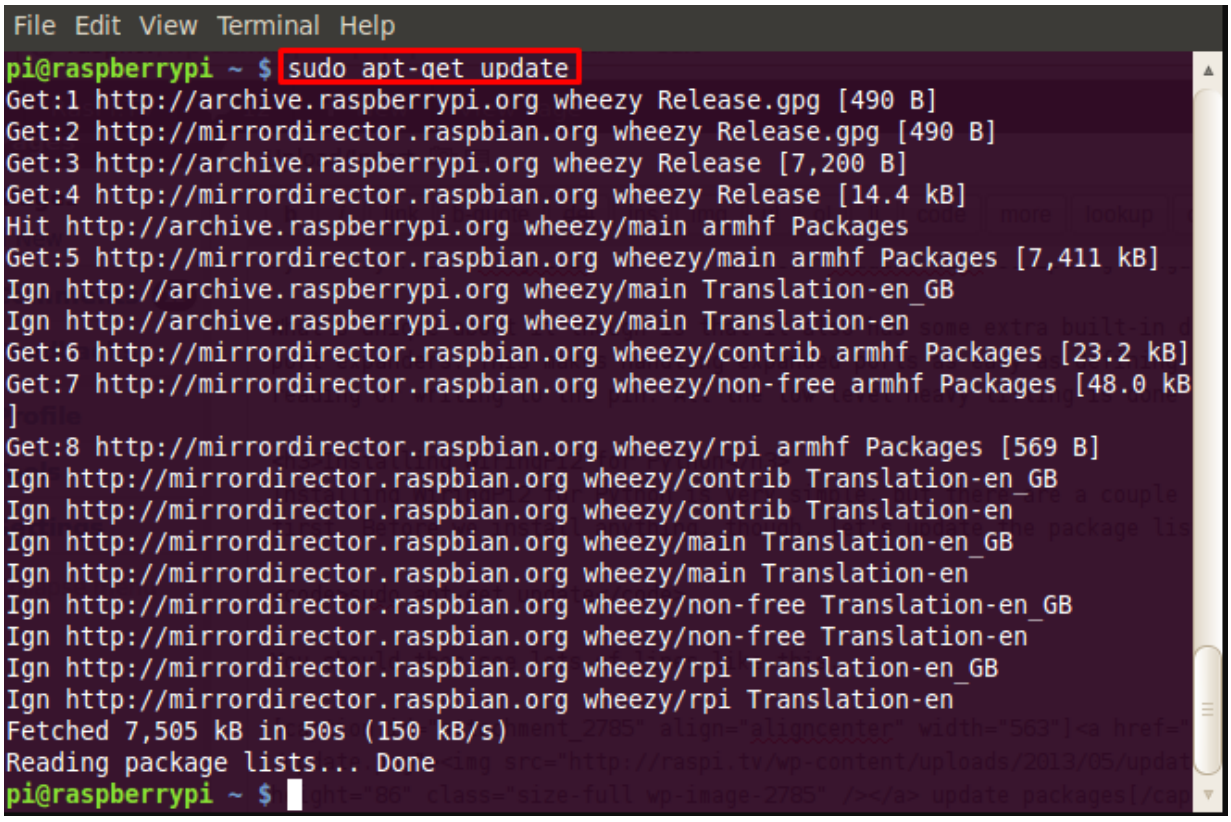


Figure 125: Update Raspbian

Install python-dev and enable GPIO, I2C:

```
sudo apt-get install python-dev

#make sure to COMMENT the lines with spi-bcm2708 and i2c-bcm2708 modules
sudo vim /etc/modprobe.d/raspi-blacklist.conf

#be sure to have i2c-dev module in the /etc/module
sudo vim /etc/modules

sudo adduser pi i2c
sudo apt-get update

#reboot:
sudo shutdown -r now

#get the GPIO software
wget https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.5.3a.tar.gz

#untar it and install it
tar -xzvf RPi.GPIO-0.5.3a.tar.gz
cd RPi.GPIO-0.5.3a/
sudo python setup.py install
sudo apt-get install i2c-tools
```

Figure 126: Install and enable GPIO on Raspbian

Finally, clone the code from the following git server and run Hub.py with python 3

https://github.com/Hinaka/-FPT-CAPSTONE-PGSS/tree/master/Raspberry%20Pi

1.3. Mobile Application

In a real product environment, user can go to play store / app store on their respective phone and search for our app to install. But for demo sake, in our current environment, user can get the apk file from the following git server:

https://github.com/Hinaka/-FPT-CAPSTONE-PGSS/tree/master/Android

2. User Guide

2.1. Manager

2.1.1. Edit car park information

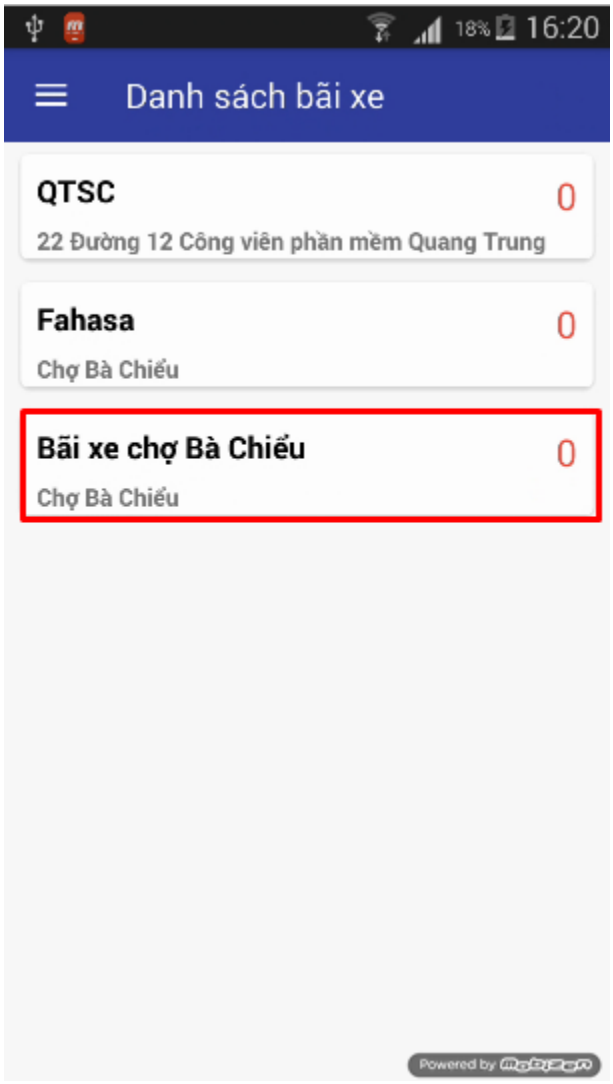


Figure 127: Edit car park information step 1

Step	Description
1	Hold/Long click the item on the list you want to edit

Table 243: Edit car park information step 1

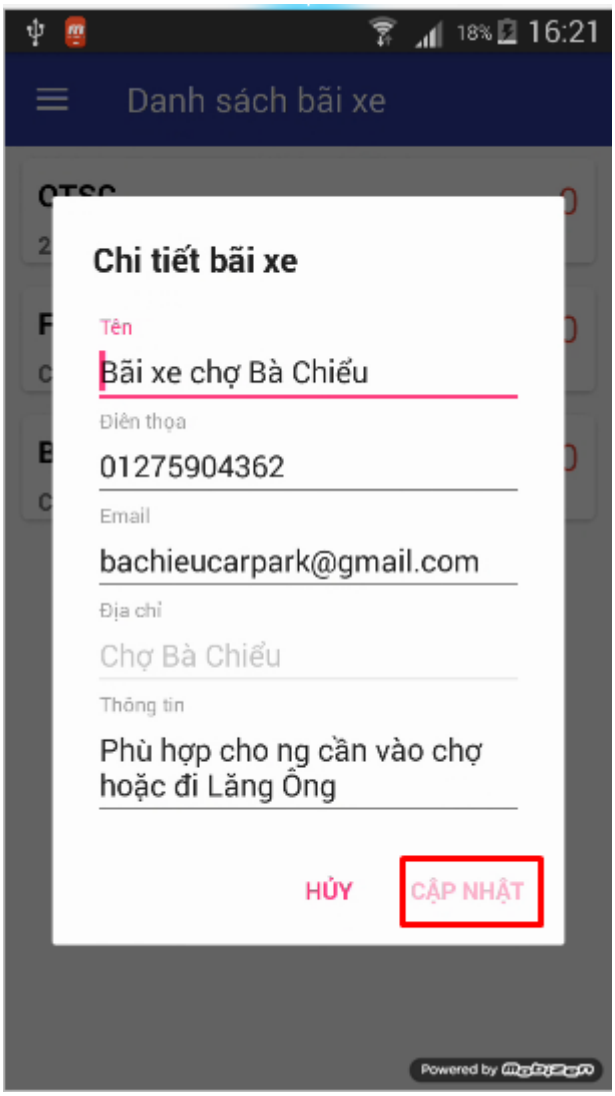


Figure 128: Edit car park information step 2

Step	Description
2	Change car park information and click "CẬP NHẬT" to finish the edit action.

Table 244: Edit car park information step 2

2.1.2. Edit area information

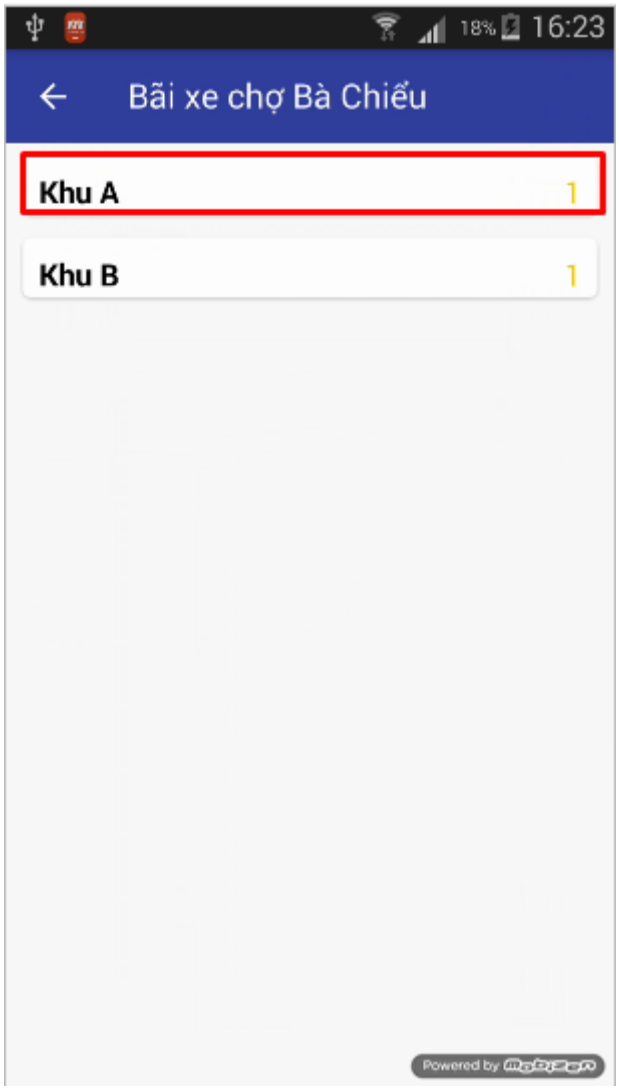


Figure 129: Edit area information step 1

Step	Description
1	Hold/Long click the item on the list you want to edit

Table 245: Edit area information step 1

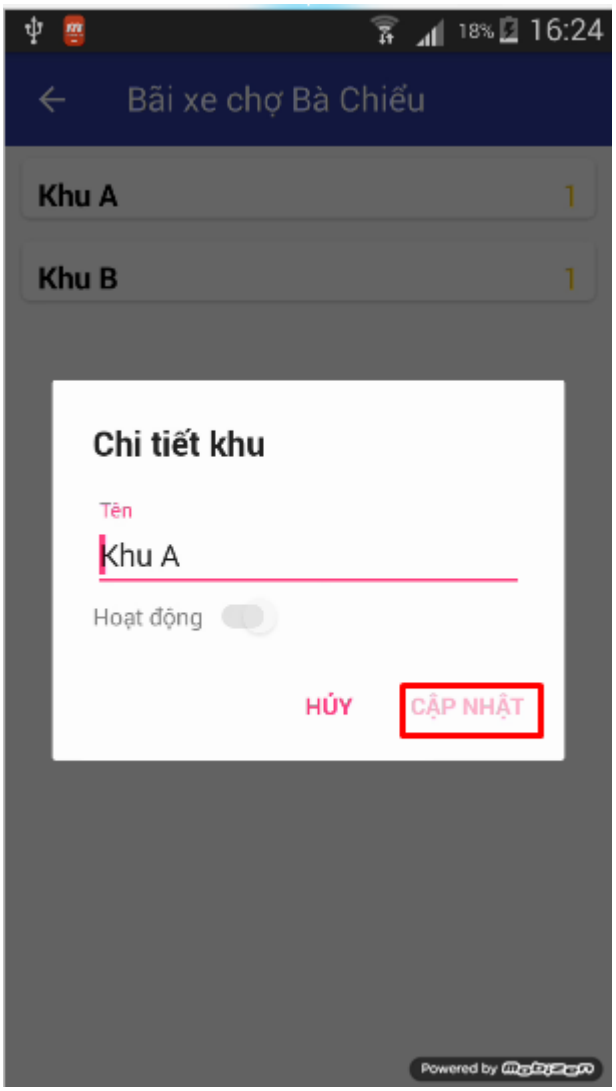


Figure 130: Edit area information step 2

Step	Description
2	Change area information and click “CẬP NHẬT” to finish the edit action.

Table 246: Edit area information step 2

2.1.3. Edit parking lot information

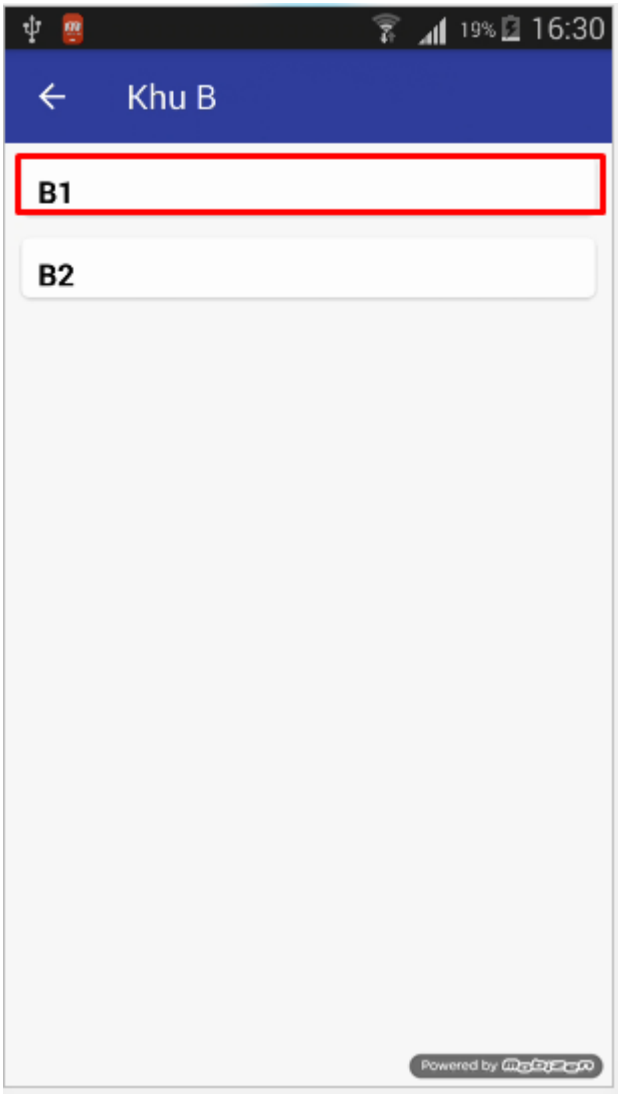


Figure 131: Edit parking lot information step 1

Step	Description
1	Hold/Long click the item on the list you want to edit

Table 247: Edit parking lot information step 1

Figure 132: Edit parking lot information step 2

Step	Description
2	Change parking lot information and click "CẬP NHẬT" to finish the edit action.

Table 248: Edit parking lot information step 2

2.1.4. Check PIN code for user

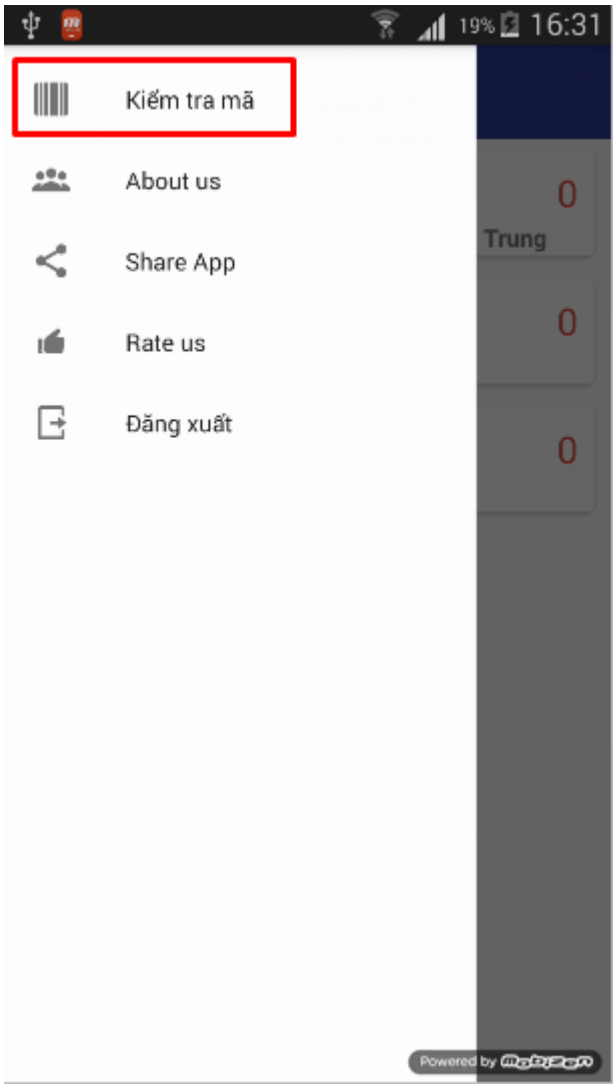


Figure 133: Check PIN code for user step 2

Step	Description
1	On main manager screen, click on the notification button
2	On notification drawer, click “Kiểm tra mã”

Table 249: Check PIN code for user step 1, 2

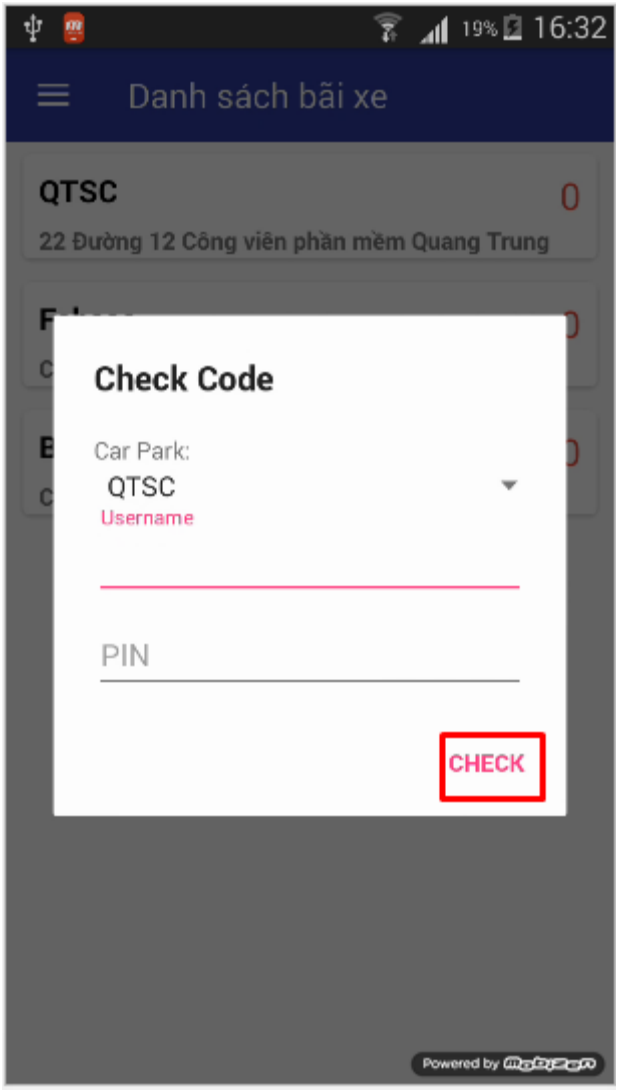


Figure 134: Check PIN code for user step 3

Step	Description
3	Manager select car park and input username, PIN code and click CHECK to finish the action. A message will show if the code is correct.

Table 250: Check PIN code for user step 3

2.2. User

2.2.1. Search for car park near user in map

Figure 135: Search for car park near user in map step 1, 2

Step	Description
1	Turn on the app and login as User or Guest
2	When the app first open or when the user clicks the Current Location button on the top right of the screen, the app will center to the current location of user. The nearest car park will be display on the map in a custom map marker like the screen above.

Table 251: Search for car park near user in map step 1, 2

Figure 136: Search for car park near user in map step 3, 4

Step	Description
3	User can drag on the map to look for other car park near current location, zoom-in, zoom-out…
4	User click on the marker, an info window will display basic information about car park for user. The basic information include: name, distance away from current location user and address.

Table 252: Search for car park near user in map step 3, 4

2.2.2. Search for car park near location in map

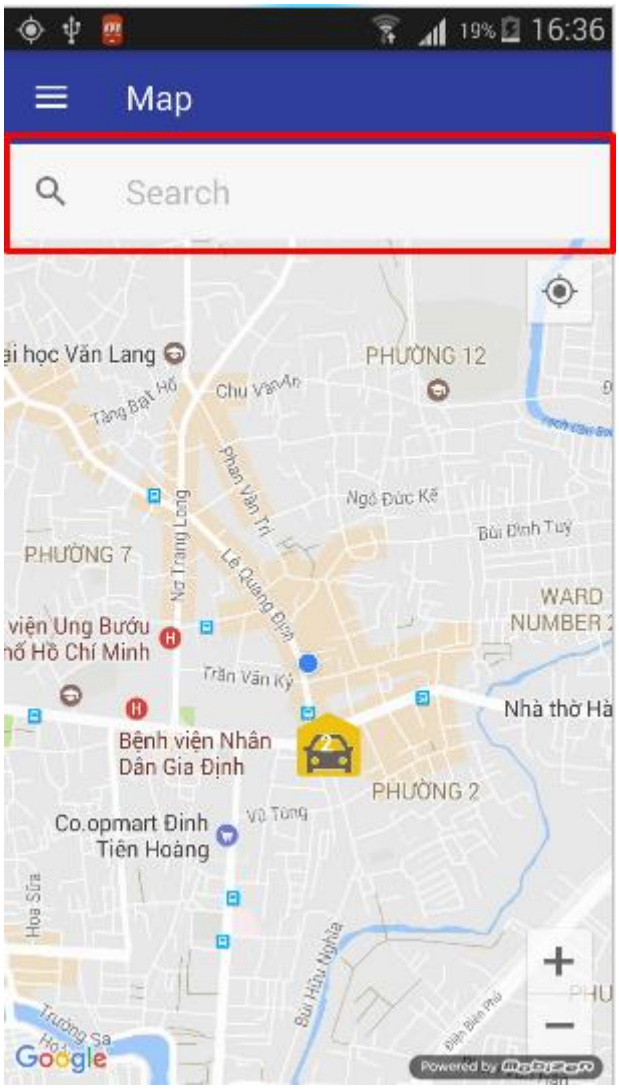


Figure 137: Search for car park near location in map step 1

Step	Description
1	User click on the search text box in map view
2	User input the name for the search location, a place drop down list will display to help user complete the input quicker.
3	User can click on the item of drop down list or click ok on keyboard.

Table 253: Search for car park near location in map step 1, 2, 3

Figure 138: Search for car park near location in map step 4

Step	Description
4	The app will center the map to searched location and display car parks near it. The value inside the search text box is the name of searched location
5	User can drag on the map to look for other car park near searched location, zoom-in, zoom-out…
6	User click on the marker, an info window will display basic information about car park for user. The basic information include: name, distance away from searched location and address.

Table 254: Search for car park near location in map step 4, 5, 6

2.2.3. Search for car park near user/location in list view

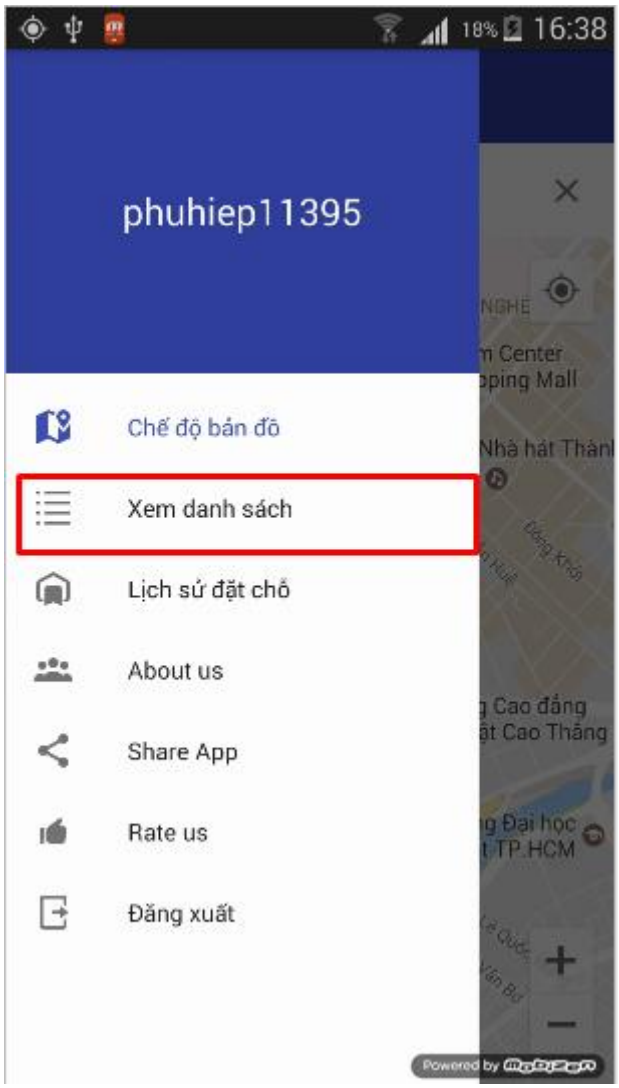


Figure 139: Search for car park near user/location in list view step 2

Step	Description
1	On map view, click on the navigation button
2	On navigation drawer, click on “Xem danh sách”

Table 255: Search for car park near user/location in list view step 1, 2

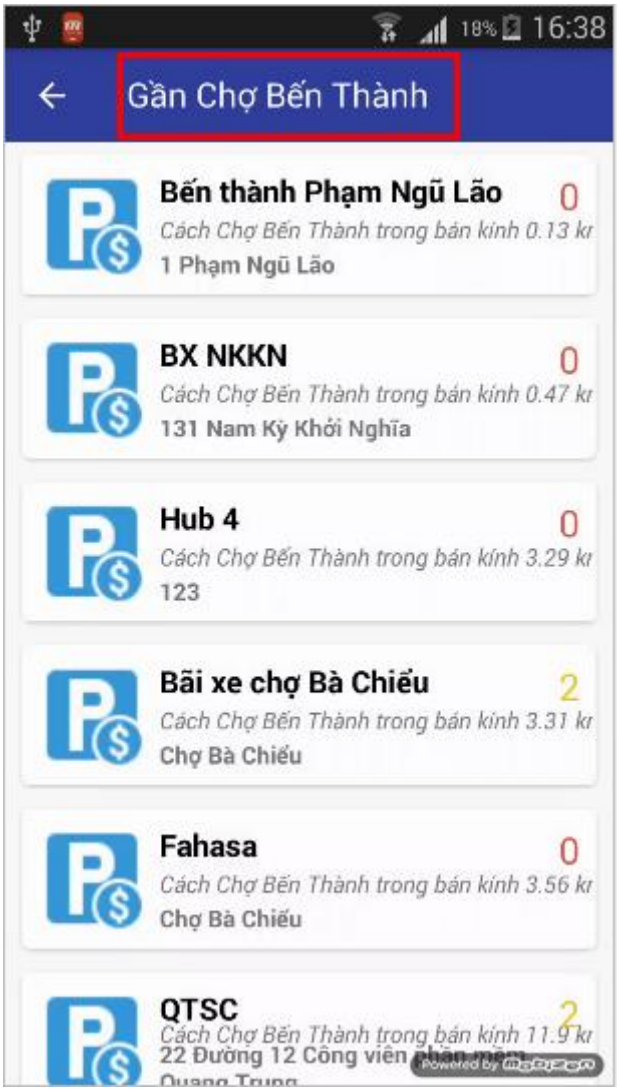


Figure 140: Search for car park near user/location in list view step 3

Step	Description
3	The app will display nearest car parks around current user location / searched location. The item in list will display basic details, include: name, distance away from user location / searched location, address.

Table 256: Search for car park near user/location in list view step 3

2.2.4. Call the car park

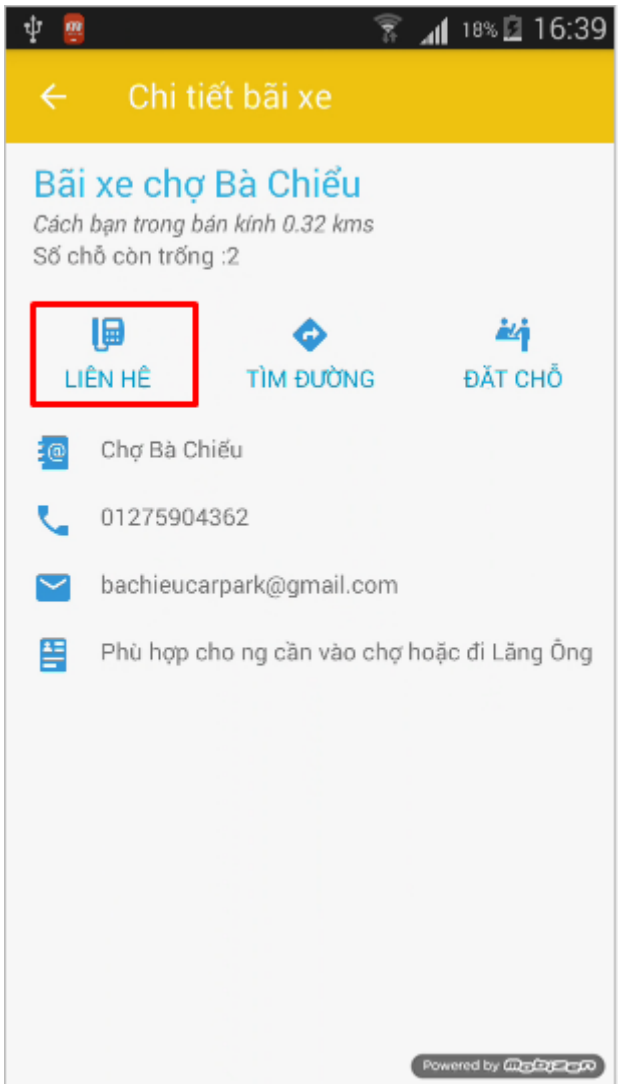


Figure 141: Call the car park step 1

Step	Description
1	On the car park detail screen, click “LIÊN HỆ”

Table 257: Call the car park step 1

Figure 142: Call the car park step 2

Step	Description
2	The app will perform the call action to car park phone number shown on the previous screen.

Table 258: Call the car park step 2

2.2.5. Find best route from current location to car park

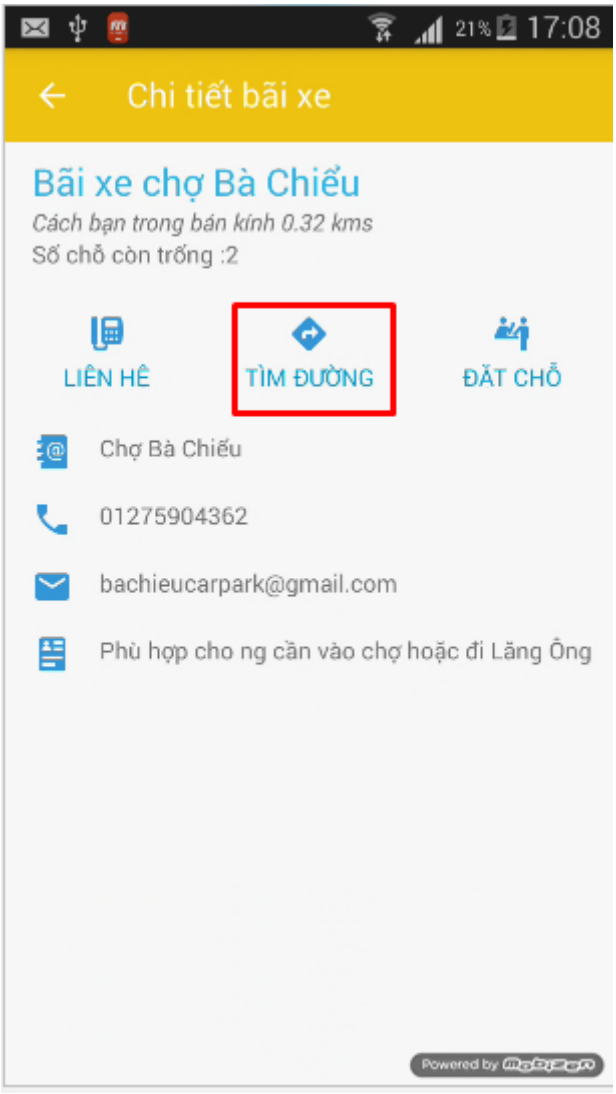


Figure 143: Find best route from current location to car park step 1

Step	Description
1	On the car park detail screen, click “TÌM ĐƯỜNG”

Table 259: Find best route from current location to car park step 1

Figure 144: Find best route from current location to car park step 2

Step	Description
2	The app will send an intent to request Google Map app to find the best route from current location to car park location

Table 260: Find best route from current location to car park step 2

2.2.6. Reserve a parking lot in car park

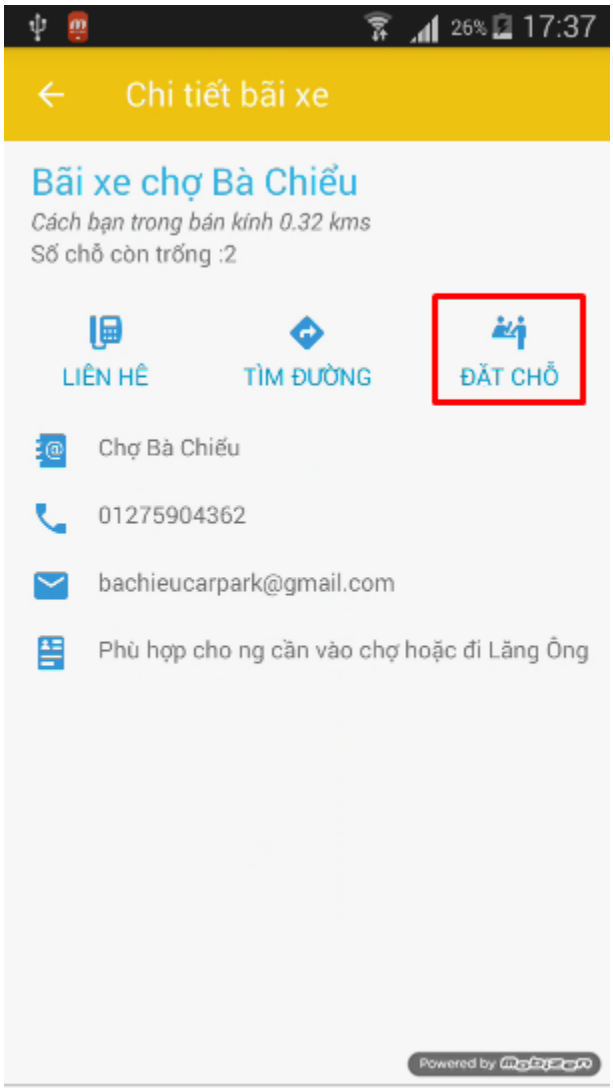


Figure 145: Reserve a parking lot in car park step 1

Step	Description
1	On car park detail screen, click “ĐẶT CHỖ”

Table 261: Reserve a parking lot in car park step 1

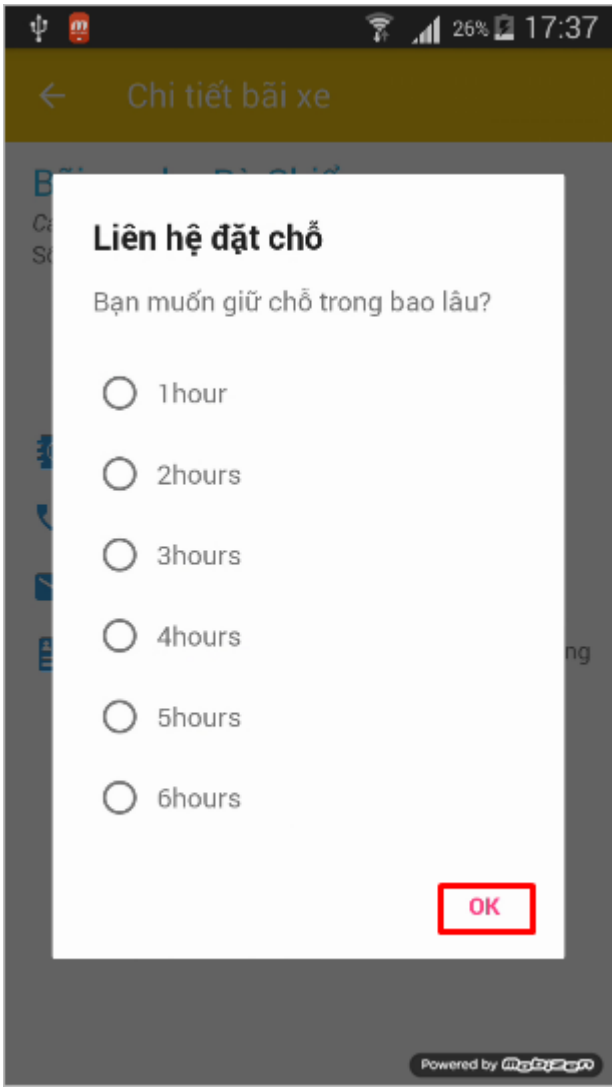


Figure 146: Reserve a car park in parking lot step 2

Step	Description
2	User will choose the duration for the reservation, and then click OK

Table 262: Reserve a car park in parking lot step 2

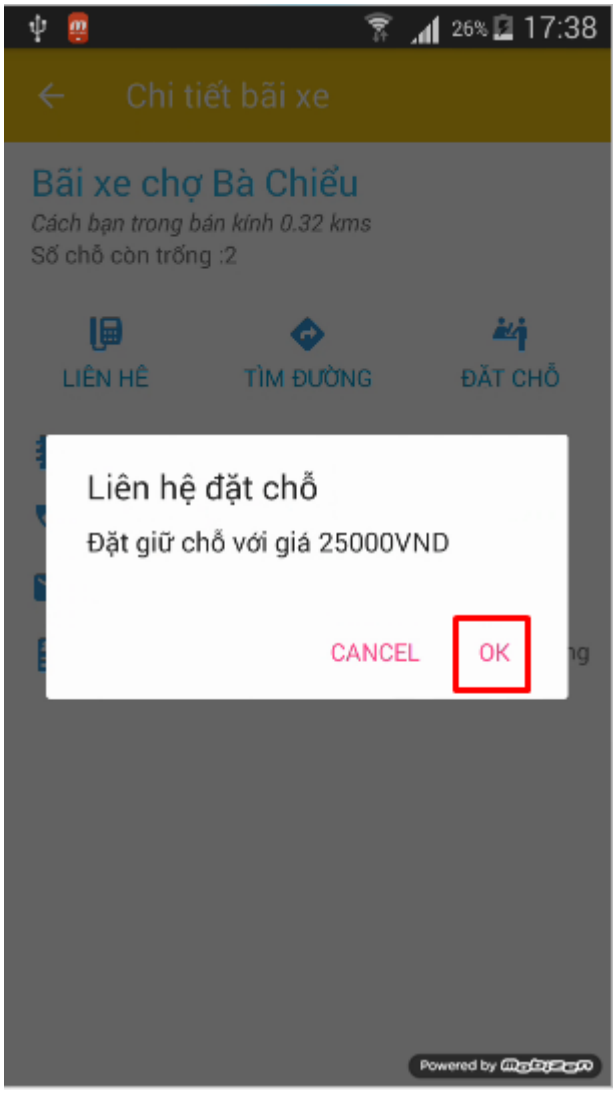


Figure 147: Reserve a parking lot in car park step 3

Step	Description
3	A dialog contains the total cost of the reservation will be display, user click OK to finish the reserve process.

Table 263: Reserve a parking lot in car park step 3

2.2.7. Cancel/Check-in reservation

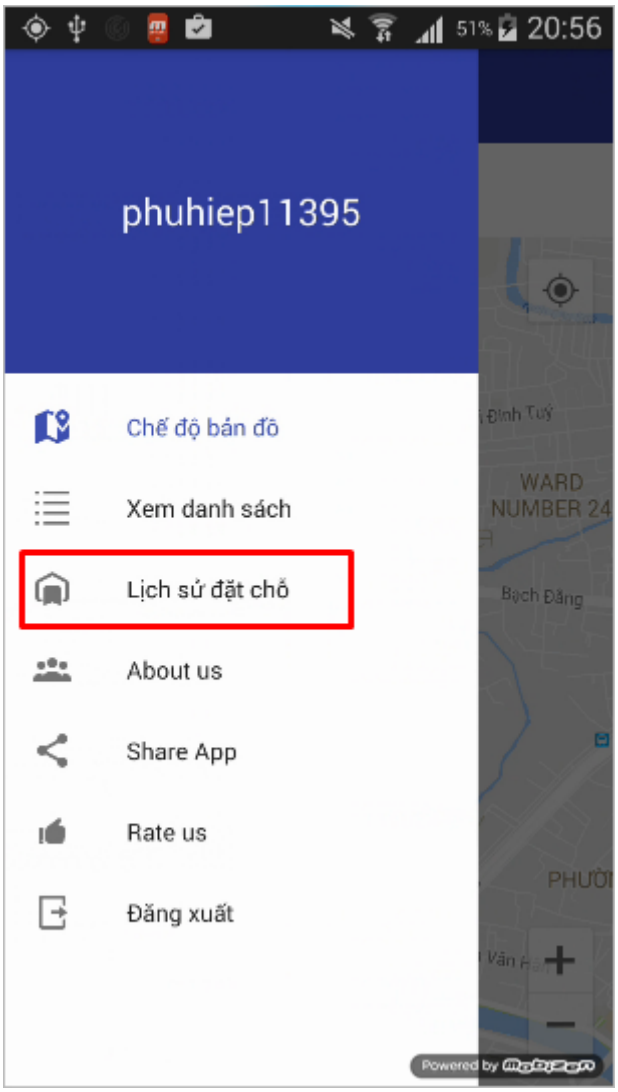


Figure 148: Cancel/Check-in reservation step 2

Step	Description
1	On map view, click on navigation button
2	On navigation drawer, click “Lịch sử đặt chỗ”

Table 264: Cancel/Check-in reservation step 1, 2

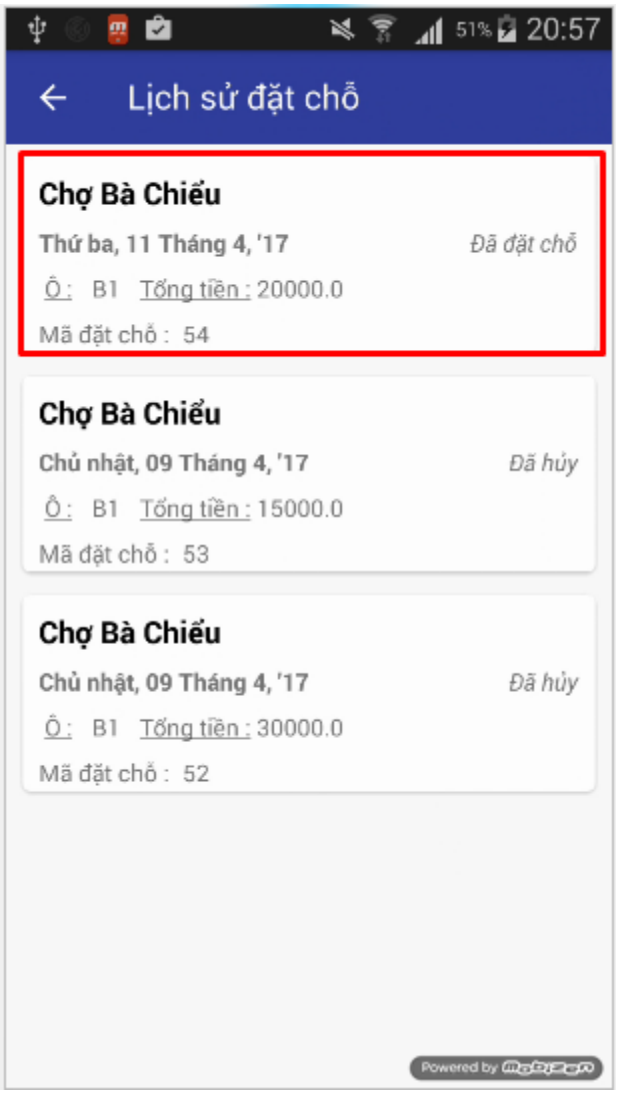


Figure 149: Cancel/Check-in reservation step 3

Step	Description
3	Select the reservation you want to cancel/check-in. Only reservation with status “Đã đặt chỗ” can be select

Table 265: Cancel/Check-in reservation step 3

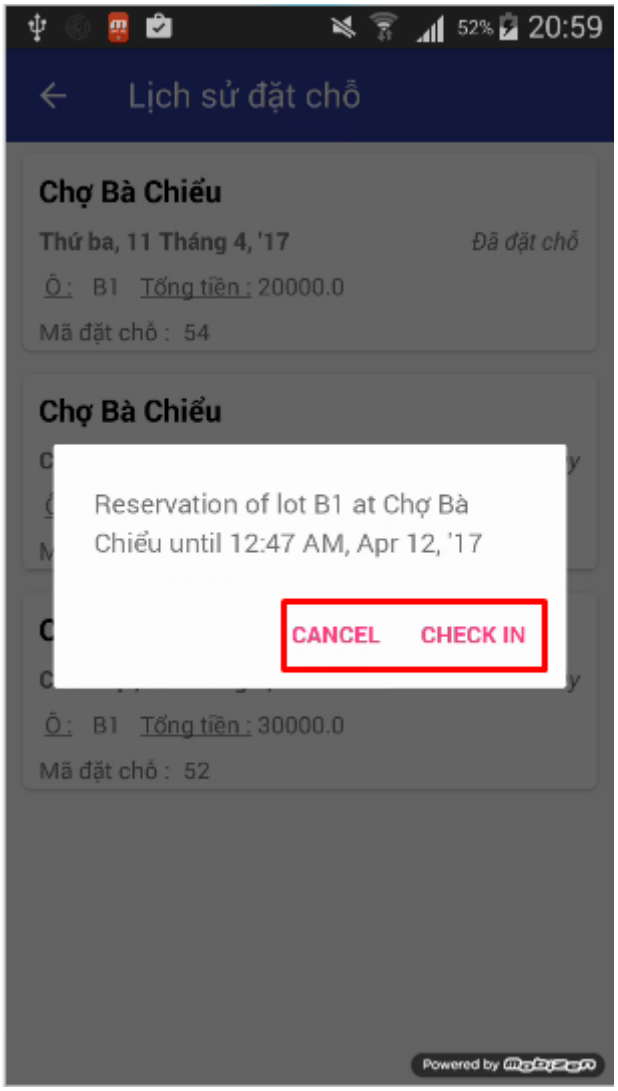


Figure 150: Cancel/Check-in reservation step 4

Step	Description
4	A dialog contains detail information of the reservation will be shown. Choose CANCEL, CHECK IN to finish the action

Table 266: Cancel/Check-in reservation step 4

G. Appendix

How to Connect a Voltage Regulator in a Circuit available at http://www.learningaboutelectronics.com/Articles/How-to-connect-a-voltage-regulator-in-a-circuit

Calculate distance from geo location available at https://msdn.microsoft.com/en-us/library/system.device.location.geocoordinate.getdistanceto(v=vs.110).aspx

Deriving the Haversine Formula available at http://mathforum.org/library/drmath/view/51879.html

Raspberry Pi 3 model B datasheet available at https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

Arduino Board Nano information available at https://www.arduino.cc/en/Main/arduinoBoardNano

nRF24L01+ datasheet available at https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

HMC5883L 3-Axis Digital Compass datasheet available at https://cdn-shop.adafruit.com/datasheets/HMC5883L_3-Axis_Digital_Compass_IC.pdf

TPIC6b595 datasheet available at http://www.ti.com/lit/ds/symlink/tpic6b595.pdf

TIP122 datasheet available at https://www.onsemi.com/pub/Collateral/TIP120-D.PDF

Principles of communications networks and systems By Nevio Benvenuto and Michele Zorzi available at http://as.wiley.com/WileyCDA/WileyTitle/productCd-0470744316.html

A painless guide to CRC Error detection algorithms By Ross N. Williams available at http://www.zlib.net/crc_v3.txt

The Internet of Things: An Overview By Karen Rose, Scott Eldridge and Lyman Chapin available at https://www.internetsociety.org/doc/iot-overview