



TRƯỜNG ĐẠI HỌC SÀI GÒN

Khoa Công nghệ thông tin

TIÊU LUẬN

ĐỒ ÁN CHUYÊN NGÀNH (841476)

ĐỀ TÀI: XÂY DỰNG ỨNG DỤNG ĐẶT VÉ
XE KHÁCH

Sinh viên thực hiện: Đặng Huỳnh Như Y

MSSV: 3120410635

Giảng viên hướng dẫn: PGS. TS. Nguyễn Tuấn Đăng

Năm học 2023 - 2024. Học kỳ 1.

TP Hồ Chí Minh, ngày 05 tháng 12 năm 2023

Nhận xét của giảng viên hướng dẫn

Lời cảm ơn.

Đầu tiên, em xin gửi lời cảm ơn chân thành đến Trường Đại học Sài Gòn đã đưa môn học Đò án chuyên ngành vào chương trình giảng dạy. Đặc biệt, em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn – Thầy Nguyễn Tuấn Đăng đã chỉ bảo, truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua. Trong thời gian tham gia lớp học Đò án chuyên ngành của thầy, chúng em đã có thêm cho mình nhiều kiến thức bổ ích, tinh thần học tập hiệu quả, nghiêm túc nhờ đó có thể tích lũy được vốn kiến thức nền tảng, giúp chúng em rất nhiều trong suốt quá trình xây dựng đồ án. Đây chắc chắn sẽ là những kiến thức quý báu, là hành trang để em có thể vững bước sau này.

Bộ môn Đò án chuyên ngành là môn học thú vị, vô cùng bổ ích và có tính thực tế cao. Đảm bảo cung cấp đủ kiến thức, gắn liền với nhu cầu thực tiễn của sinh viên. Tuy nhiên, do vốn kiến thức còn nhiều hạn chế và khả năng tiếp thu thực tế còn nhiều bỡ ngỡ. Mặc dù em đã cố gắng hết sức nhưng chắc chắn bài đồ án khó có thể tránh khỏi những thiếu sót và nhiều chỗ còn chưa chính xác, kính mong thầy xem xét và góp ý để bài Đò Án của em được hoàn thiện hơn.

Em xin chân thành cảm ơn.

Lời mở đầu.

Trong thời đại công nghệ ngày nay, việc ứng dụng các phương tiện tiện ích để giải quyết nhu cầu hàng ngày ngày càng trở nên quan trọng và tối ưu hóa. Đặc biệt, lĩnh vực giao thông và vận tải không chỉ là một yếu tố quan trọng góp phần vào sự phát triển của mỗi quốc gia mà còn ảnh hưởng trực tiếp đến chất lượng cuộc sống của cộng đồng. Trong bối cảnh này, việc xây dựng và phát triển ứng dụng đặt vé xe khách trở thành một xu hướng không thể phủ nhận.

Thị trường Việt Nam, với sự tăng trưởng nhanh chóng của ngành công nghiệp và sự phát triển không ngừng của hạ tầng giao thông, đặt ra nhiều thách thức và cơ hội cho việc cải thiện hệ thống vận tải công cộng. Trong bối cảnh đó, đồ án này hướng đến mục tiêu xây dựng một ứng dụng đặt vé xe khách hiệu quả, linh hoạt và thuận tiện nhằm nâng cao trải nghiệm người sử dụng và đồng thời đóng góp vào sự hiện đại hóa trong lĩnh vực vận tải.

Ứng dụng đặt vé xe khách không chỉ giúp người dùng tiết kiệm thời gian mà còn mang lại lợi ích cho khách hàng, thúc đẩy sự tiện lợi trong ngành vận tải đường bộ. Đáp ứng được nhu cầu đặt vé xe khách đi các tỉnh thành khác nhau. Tăng tính hiệu quả trong việc đặt vé xe thời gian và công sức đặt vé. Ứng dụng cung cấp thông tin các nhà xe, các tuyến đường và các giá vé đi các tỉnh thành khác nhau để người dùng có thể đặt vé dễ dàng.

Em tin rằng việc xây dựng ứng dụng đặt vé xe khách không chỉ là một bước tiến quan trọng mà còn là sự đóng góp tích cực cho sự phát triển bền vững của đất nước. Hy vọng rằng đồ án này sẽ mang lại những giải pháp sáng tạo và hiệu quả, góp phần vào việc thúc đẩy tiện ích và chất lượng cuộc sống cho cộng đồng.

MỤC LỤC

Mục lục bảng.....	1
Mục lục hình ảnh.....	3
Chương 1: Tổng quan về đề tài.....	8
1.1. Đề tài.....	8
1.1.1. Tên đề tài: Xây dựng ứng dụng đặt vé xe khách	8
1.1.2. Mục tiêu.....	8
1.1.3. Đối tượng nghiên cứu:	8
1.1.4. Phạm vi:	10
1.1.5. Đóng góp mới (dự kiến)	10
1.2. Kế hoạch triển khai.....	11
1.3. Khảo sát tổng quan.	11
1.3.1. Lịch sử.....	11
1.3.2. Các công trình có liên quan.....	12
Chương 2: Xác định thu thập yêu cầu	13
2.1. Tổng quan về ứng dụng.	13
2.2. Khảo sát hiện trạng nghiệp vụ.	13
2.3. Xác định yêu cầu.	13
2.3.1. Yêu cầu chức năng.	13
2.3.2. Yêu cầu phi chức năng.....	14
2.3.3. BRD	14
2.3.3.1. Tóm tắt dự án.	14
2.3.3.2. Mục tiêu.	14
2.3.3.3. BRD chi tiết.	14
2.3.3.3.1. Chức năng đăng ký tài khoản.....	14
2.3.3.3.2. BRD chức năng đăng nhập.....	17

2.3.3.3.3. BRD chức năng đặt vé xe	19
2.3.4.4. BRD chức năng xem lịch sử đặt vé.....	22
2.3.4. PRD.....	25
2.3.4.1. PRD chức năng đăng ký.	25
2.3.4.2. PRD chức năng đăng nhập.	27
2.3.4.3. PRD chức năng đặt vé xe.....	29
2.4.3.3.1. PRD chức năng chọn tuyến đường.....	29
2.4.3.3.2. PRD chức năng chọn nhà xe.	30
2.4.3.3.3. PRD chức năng chọn chỗ ngồi.....	33
2.4.3.3.4. PRD chức năng chọn điểm đón/trả.	35
2.4.3.3.5. PRD chức năng nhập thông tin liên hệ.	37
2.4.3.3.6. PRD hiển thị thông tin đặt vé.	39
2.3.4.7. PRD chức năng xem lịch sử đặt vé xe.....	41
2.3.4.7.1. PRD danh sách lịch sử vé xe.	41
2.3.4.7.1. PRD danh sách lịch sử vé xe.	42
Chương 3: Phân tích thiết kế	44
3.1. Use case.....	44
3.1.1. Use case tổng quát.....	44
3.1.2. Phân rã và đặc tả use case.....	46
3.1.2.1. Use case đăng ký.....	46
3.1.2.2. Use case đăng nhập.....	48
3.1.2.3. Use case đăng xuất.....	50
3.1.2.4. Use case đặt vé.....	51
3.1.2.5. Use case xem lịch sử.....	52
3.2. Sơ đồ phân rã chức năng BFD	54
3.3. Sơ đồ DFD	55

3.3.1. Sơ đồ DFD tổng quát:	55
3.3.2. Sơ đồ DFD chi tiết:	56
3.3.2.1. Sơ đồ DFD chức năng đăng nhập:.....	56
3.3.2.2. Sơ đồ DFD chức năng đăng ký:.....	58
3.3.2.3. Sơ đồ DFD chức năng đặt vé xe:.....	58
3.4. ERD.....	62
3.4.1. ERD giữa Customer và BookingSeat.	62
3.4.2. ERD giữa BookingSeat, BookingSeatDetail và Carriers.	63
4.4.3. ERD giữa Province, Routes và Carriers.	63
3.4.4. ERD giữa Chair, Trip và Routes.....	64
3.5. Activity.....	66
3.5.1. Activity đăng ký.....	66
3.5.2. Activity đăng nhập.	67
3.5.3. Activity đặt vé	68
3.5.4. Activity xem lịch sử.....	69
3.6. Biểu đồ lớp	70
Chương 4: Thiết kế phần mềm	71
4.1. Thiết kế giao diện.	71
4.1.1. Thiết kế giao diện đăng nhập.....	71
4.1.2. Thiết kế giao diện đăng ký.	73
4.1.3. Thiết kế giao diện đặt vé.....	76
4.1.3.1. Thiết kế giao diện chọn tuyến đường	76
4.1.3.2. Thiết kế giao diện chọn nhà xe	78
4.1.3.3. Thiết kế giao diện thông tin nhà xe	82
4.1.3.4. Thiết kế giao diện chọn ghế.....	85
4.1.3.5. Thiết kế giao diện chọn điểm đón/điểm trả	89

4.1.3.6. Thiết kế giao diện thông tin liên hệ	91
4.1.3.7. Thiết kế giao diện thông tin đặt vé.	94
4.1.4. Thiết kế giao diện lịch sử đặt vé.....	97
4.1.5. Thiết kế giao diện tài khoản.....	99
Chương 5: Viết API và deploy lên server.	101
5.1. Cài đặt và tạo dự án NodeJS trên Windows 11.....	101
5.2. Viết request register.....	104
5.3. Viết request login.....	107
5.4. MongoDB và kết nối với NodeJS.....	113
5.5. Viết request get routes province.....	121
5.6. Viết request update routes.....	123
5.7. Viết request insert booking seat.....	125
5.8. Viết request get booking seat by id customer.....	127
5.9. Deploy API lên Cyclic	128
5.10. Link githup API.....	131
Chương 6: Lập trình và kiểm thử	132
6.1. Lập trình	132
6.1.1. Cấu trúc các thư mục.....	132
6.1.2. Màn hình đăng nhập.....	137
6.1.3. Màn hình đăng ký.....	139
6.1.4. Màn hình chọn tuyến đường.....	141
6.1.5. Màn hình chọn nhà xe.	143
6.1.6. Màn hình chọn chỗ ngồi.	145
6.1.7. Màn hình chọn điểm đón, điểm trả.	147
6.1.8. Màn hình nhập thông tin liên hệ.	149
6.1.9. Màn hình hiển thị dữ liệu thông tin đặt vé.....	150

6.1.10. Màn hình lịch sử đặt vé.	153
6.1.11. Màn hình hiển thị thông tin chi tiết lịch sử đặt vé.	155
6.1.12. Màn hình tài khoản.	157
6.2. Kiểm thử.....	159
Chương 7: Kết luận và hướng mở rộng.....	160
7.1. Kết quả thu được.....	160
7.2. Hướng phát triển trong tương lai.	160
7.3. Kết luận.	160
7.4. Link source code:	160
Tài liệu tham khảo.....	161

Mục lục bảng.

Bảng 1. Kế hoạch triển khai dự án	11
Bảng 2. Mô tả actor	44
Bảng 3. Mô tả các usecase	45
Bảng 4. Đặc tả use case đăng ký	47
Bảng 5. Đặc tả use case đăng nhập.	49
Bảng 6. Đặc tả use case đăng xuất.	50
Bảng 7. Đặc tả use case đặt vé.	52
Bảng 8. Đặc tả use case xem lịch sử đặt vé.	54
Bảng 9. Biến cõ giao diện đăng nhập.....	72
Bảng 10. Mô tả giao diện đăng nhập.....	73
Bảng 11. Biến cõ giao diện đăng ký.....	74
Bảng 12. Mô tả giao diện đăng ký.	75
Bảng 13. Biến cõ giao diện chọn tuyến đường.	77
Bảng 14. Mô tả giao diện chọn tuyến đường.	78
Bảng 15. Biến cõ giao diện chọn nhà xe.....	80
Bảng 16. Mô tả giao diện chọn nhà xe.....	81
Bảng 17. Biến cõ giao diện thông tin nhà xe.	83
Bảng 18. Mô tả giao diện thông tin nhà xe.	84
Bảng 19. Biến cõ giao diện chọn ghế.....	86
Bảng 20. Mô tả giao diện chọn ghế.....	88
Bảng 21. Biến cõ giao diện chọn điểm đón, điểm trả.	90
Bảng 22. Mô tả giao diện chọn điểm đón, điểm trả.	91
Bảng 23. Biến cõ giao diện thông tin liên hệ.	93
Bảng 24. Mô tả giao diện thông tin liên hệ.....	93
Bảng 25. Biến cõ giao diện thông tin đặt vé.	95
Bảng 26. Mô tả giao diện thông tin đặt vé.	96
Bảng 27. Biến cõ lịch sử đặt vé.....	98
Bảng 28. Mô tả giao diện lịch sử đặt vé.....	98

Bảng 29. Biến cố giao diện tài khoản.....	100
Bảng 30. mô tả giao diện tài khoản.....	100
Bảng 31. Kiểm thử	159

Mục lục hình ảnh.

Hình 1. Workflow chức năng đăng ký	16
Hình 2. Workflow chức năng đăng nhập.....	18
Hình 3. Workflow chức năng đặt vé xe.....	21
Hình 4. Workflow chức năng xem lịch sử đặt vé.....	23
Hình 5. UI mockup đăng ký.....	26
Hình 6. UI mockup đăng nhập.	28
Hình 7. UI mockup chọn tuyến đường.....	29
Hình 8. UI mockup chọn nhà xe.	31
Hình 9. UI mockup thông tin nhà xe.....	32
Hình 10. UI mockup chọn ghế ngồi.....	34
Hình 11. UI mockup chọn điểm đón, điểm trả.....	36
Hình 12. UI moc kup thông tin liên hệ.	38
Hình 13. UI mockup thông tin đặt vé.....	39
Hình 14. UI mockup lịch sử đặt vé.	41
Hình 15. UI mockup chi tiết lịch sử đặt vé.	42
Hình 16. Use case tổng quát.....	44
Hình 17. Use case đăng ký.....	46
Hình 18. Use case đăng nhập.....	48
Hình 19. Use case đăng xuất.....	50
Hình 20. Use case đặt vé.	51
Hình 21. Use case xem lịch sử đặt vé.	53
Hình 22. Sơ đồ phân rã chức năng BFD.	54
Hình 23. Sơ đồ DFD tổng quát mức 0.	55
Hình 24. Sơ đồ DFD tổng quát mức 1.	55
Hình 25. Sơ đồ DFD tổng quát mức 2.	56
Hình 26. Sơ đồ DFD chức năng đăng nhập mức 0.	56
Hình 27. Sơ đồ chức năng đăng nhập mức 1.	57
Hình 28. Sơ đồ chức năng đăng nhập khách hàng mức 2.....	57

Hình 29. Sơ đồ DFD chức năng đăng ký mức 0.....	58
Hình 30. Sơ đồ chức năng đăng ký mức 1.....	58
Hình 31. Sơ đồ DFD chức năng đặt vé mức 1.....	59
Hình 32. Sơ đồ DFD chức năng đặt vé mức 1.....	59
Hình 33. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.1.....	60
Hình 34. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.2.....	60
Hình 35. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.3.....	60
Hình 36. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.4.....	61
Hình 37. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.5.....	61
Hình 38. ERD tổng quát.....	62
Hình 39. ERD giữa Customer và BookingSeat.....	62
Hình 40. ERD giữa BookingSeat, BookingSeatDetail và Carriers.....	63
Hình 41. ERD giữa Province, Routes và Carriers.....	63
Hình 42. ERD giữa Chair, Trip và Routes.	64
Hình 43. Activity đăng ký.	66
Hình 44. Activity đăng nhập.	67
Hình 45. Activity đặt vé.	68
Hình 46. Activity xem lịch sử đặt vé.....	69
Hình 47. Biểu đồ lớp.	70
Hình 48. Thiết kế giao diện đăng nhập.	71
Hình 49. Thiết kế giao diện đăng ký.	73
Hình 50. Thiết kế giao diện chọn tuyến đường.....	76
Hình 51. Thiết kế giao diện chọn nhà xe.	79
Hình 52. Thiết kế giao diện chọn nhà xe.	82
Hình 53. Thiết kế giao diện chọn chỗ ngồi.	85
Hình 54. Thiết kế giao diện chọn điểm đón, điểm trả.....	89
Hình 55. Thiết kế giao diện thông tin liên hệ.....	92
Hình 56. Thiết kế giao diện thông tin đặt vé.....	94
Hình 57. Thiết kế giao diện xem lịch sử đặt vé.	97

Hình 58. Thiết kế giao diện tài khoản.....	99
Hình 59. Cấu trúc các thư mục viết API.....	101
Hình 60. Tạo thư mục routes.....	102
Hình 61. Tạo file index.js để quản lý các file routes.....	103
Hình 62. Viết request register trong file user nằm trong thư mục routes.....	104
Hình 63. Viết hàm register trong file user.js nằm trong thư mục repositories..	104
Hình 64. Viết hàm register trong file user.js nằm trong thư mục controller.....	105
Hình 65. Tạo file auth.js nằm trong thư mục authentication.....	106
Hình 66. Viết request login trong file user nằm trong thư mục routes.	107
Hình 67. Tạo file user nằm trong controller.....	108
Hình 68. Tạo một file index.js trong controller để quản lý các file controller khác. ..	108
Hình 69. Tạo file user.js nằm trong thư mục repositories.....	109
Hình 70. Tạo một file index.js để quản lý các file repositories khác.....	109
Hình 71. Tạo file database.js để kết nối với cơ sở dữ liệu.....	110
Hình 72. Tạo một file print.js để in ra màn hình màu cho lỗi và cảnh báo, thành công.	111
Hình 73. Tạo file HttpStatusCode.js để trả về các status code khi gửi request.	111
Hình 74. Viết model routes.js.....	121
Hình 75. Viết hàm get Routes Province trong file routes.js nằm trong thư mục repositories.	121
Hình 76. Viết hàm get routes province nằm trong file routes.js nằm trong thư mục controllers.....	122
Hình 77. Viết request get routes province trong file routes.js nằm trong thư mục routes.	122
Hình 78. Viết hàm update trong file routes.js nằm trong thư mục repositories.	123
Hình 79. Viết hàm update routes trong file routes.js nằm trong thư mục controller.....	123

Hình 80. Viết request update routes trong file routes.js nằm trong thư mục routes.....	124
Hình 81. Viết hàm insert booking seat trong file bookingSeat.js nằm trong thư mục repositories.....	125
Hình 82. Viết hàm insert booking seat trong file bookingSeat.js nằm trong thư mục controllers.....	126
Hình 83. Viết request insert booking seat trong file bookingSeat.js nằm trong thư mục routes.....	126
Hình 84. Viết hàm get booking seat by customer Id trong file boolingSeat.js nằm trong thư mục repositories.....	127
Hình 85. Viết hàm get booking seat by customer id trong file booking.js nằm trong thư mục controller.....	127
Hình 86. Viết request get booking seat by customer id trong routes.....	127
Hình 87. Cấu trúc các thư mục.....	132
Hình 88. Thư mục api.....	132
Hình 89. Thư mục models.....	133
Hình 90. Thư mục presenter.....	133
Hình 91. Thư mục response.....	133
Hình 92. Thư mục SharedPreferences.....	134
Hình 93. Thư mục viewmodels	134
Hình 94. Thư mục view.....	135
Hình 95. Thư mục layout các file xml	136
Hình 96. Màn hình đăng nhập.....	137
Hình 97. Hàm đăng nhập.	138
Hình 98. màn hình đăng ký.....	139
Hình 99. Hàm đăng ký.	140
Hình 100. Màn hình chọn tuyến đường.	141
Hình 101. Hàm hiển thị ngày.	142
Hình 102. Màn hình chọn nhà xe.....	143

Hình 103. Hàm thực hiện call api để lấy dữ liệu nhà xe theo ngày đi, điểm đi, điểm đến.....	144
Hình 104. Màn hình chọn chỗ ngồi.....	145
Hình 105. Xử lý dữ liệu để đổ dữ liệu vào adapter hiển thị trạng thái ghế lên giao diện.....	146
Hình 106. Xử lý chọn và bỏ chọn ghế.....	146
Hình 107. Màn hình chọn điểm đón, điểm trả.....	147
Hình 108. Xử lý hiển thị điểm đón điểm trả trên 2 Tab View.....	148
Hình 109. Màn hình nhập thông tin liên hệ.....	149
Hình 110. Màn hình hiển thị dữ liệu thông tin đặt vé.....	150
Hình 111. Hàm updateSeat.....	151
Hình 112. Hàm InsertBookingSeat.....	152
Hình 113. Màn hình lịch sử đặt vé.....	153
Hình 114. Hàm lấy dữ liệu từ api lịch sử đặt vé.....	154
Hình 115. màn hình chi tiết lịch sử đặt vé.....	155
Hình 116. Xử lý hiển thị chi tiết thông tin lịch sử đặt vé.....	156
Hình 117. Màn hình tài khoản.....	157
Hình 118. Xử lý hiển thị tên người dùng và đăng xuất.....	158

Chương 1: Tổng quan về đề tài.

1.1. Đề tài

1.1.1. Tên đề tài: Xây dựng ứng dụng đặt vé xe khách

1.1.2. Mục tiêu

Tổng quan:

Tạo ra một ứng dụng di động mang lại lợi ích cho khách hàng, thúc đẩy sự tiện lợi trong ngành vận tải đường bộ.

Đáp ứng được nhu cầu đặt vé xe khách đi các tỉnh thành khác nhau, cho phép người dùng tìm hiểu, so sánh và đặt vé xe khách một cách nhanh chóng và tiện lợi.

Tăng tính hiệu quả trong việc đặt vé xe thời gian và công sức đặt vé. Ứng dụng cung cấp thông tin các nhà xe, các tuyến đường và các giá vé đi các tỉnh thành khác nhau để người dùng có thể đặt vé dễ dàng.

Cụ thể:

Phát triển các chức năng đặt vé cho ứng dụng, các tính năng tìm kiếm, chọn tuyến đường, lịch trình, ghế ngồi.

Xây dựng cơ sở dữ liệu chứa thông tin về các nhà xe, các tuyến đường, giá vé, lịch trình.

Ứng dụng sẽ hướng đến thị trường khách hàng có nhu cầu đi lại trong nước.

Chức năng xem lịch sử đặt vé xe.

1.1.3. Đối tượng nghiên cứu:

Java:

Theo Java là gì? Tổng quan về ngôn ngữ lập trình java [1], Java là một trong những ngôn ngữ lập trình hướng đối tượng. Nó được sử dụng trong phát triển phần mềm, trang web, game hay ứng dụng trên các thiết bị di động.

MongoDB:

Theo Bắt đầu với NoSQL và MongoDB [2], MongoDB là một cơ sở dữ liệu mã nguồn mở và là cơ sở dữ liệu NoSQL hàng đầu, được hàng triệu người sử dụng. MongoDB được viết bằng C++.

MongoDB là một cơ sở dữ liệu đa nền tảng, hoạt động trên các khái niệm Collection và Document, nó cung cấp hiệu suất cao, tính khả dụng cao và khả năng mở rộng dễ dàng.

Node JS:

Theo NodeJS là gì? Đặc điểm và ứng dụng của Node.JS [3], NodeJS là mã nguồn mở chạy trên môi trường V8 JavaScript runtime (một trình thông dịch JavaScript chạy cực nhanh trên trình duyệt Chrome).

Node.JS có thể được dùng để xây dựng các loại ứng dụng khác nhau như các ứng dụng dòng lệnh, ứng dụng web, ứng dụng trò chuyện theo thời gian thực, máy chủ REST API,.. Tuy nhiên, NodeJS thường được dùng chủ yếu để xây dựng các chương trình mạng như máy chủ web, tương tự như PHP, Java hoặc ASP.NET.

Restful API:

Theo RESTful API là gì? [4], RESTful API là một tiêu chuẩn dùng trong việc thiết kế API cho các ứng dụng web (thiết kế Web services) để tiện cho việc quản lý các resource. Nó chú trọng vào tài nguyên hệ thống (tệp văn bản, ảnh, âm thanh, video, hoặc dữ liệu động...), bao gồm các trạng thái tài nguyên được định dạng và được truyền tải qua HTTP.

API (Application Programming Interface) là một tập các quy tắc và cơ chế mà theo đó, một ứng dụng hay một thành phần sẽ tương tác với một ứng dụng hay thành phần khác. API có thể trả về dữ liệu mà bạn cần cho ứng dụng của mình ở những kiểu dữ liệu phổ biến như JSON hay XML.

REST (REpresentational State Transfer) là một dạng chuyển đổi cấu trúc dữ liệu, một kiểu kiến trúc để viết API. Nó sử dụng phương thức HTTP đơn giản để tạo cho giao tiếp giữa các máy. Vì vậy, thay vì sử dụng một URL cho việc xử lý một số thông tin người dùng, REST gửi một yêu cầu HTTP như GET, POST, DELETE đến một URL để xử lý dữ liệu.

Retrofit:

Theo Retrofit là gì và cách sử dụng - Caching dữ liệu với Retrofit - Lưu dữ liệu offline [5], được phát triển bởi Square

Retrofit là một HTTP client type-safe cho Android và Java. Retrofit giúp dễ dàng kết nối đến một dịch vụ REST trên web bằng cách chuyển đổi API thành Java Interface.

Retrofit rất mạnh mẽ giúp bạn dễ dàng xử lý dữ liệu JSON hoặc XML sau đó phân tích cú pháp thành Plain Old Java Objects (POJOs). Tất cả các yêu cầu GET, POST, PUT, PATCH, và DELETE đều có thể được thực thi.

Kiến trúc MVVM trong android:

Theo Kiến trúc dự án trong Android – MVVM với Data Binding [6], view: Bind tới các biến observable và các sự kiện của ViewModel một cách linh hoạt. View bao gồm các tệp .xml và các Activity/Fragment.

Theo Kiến trúc dự án trong Android – MVVM với Data Binding [6], model: Tầng dữ liệu, chịu trách nhiệm quản lý business login, data, state. Trong Android đối tượng mô tả dữ liệu như các Class, hàm xử lý,...

Theo Kiến trúc dự án trong Android – MVVM với Data Binding [6], ViewModel: ViewModel chịu trách nhiệm cập nhật dữ liệu cho Model và chuẩn bị dữ liệu Observable cần thiết cho View. Tuy nhiên ViewModel không gắn với View.

Cyclic:

Theo Deploying a NestJS app for Free on Cyclic [7], cyclic là nền tảng đám mây cho phép các lập trình viên xây dựng, triển khai, quản lý và mở rộng ứng dụng

1.1.4. Phạm vi:

Môi trường ứng dụng mobile (android)

Deploy API lên server (Cyclic)

1.1.5. Đóng góp mới (dự kiến)

Phần lớn những app hiện nay chỉ quan tâm đến việc đặt xe của khách hàng. Không quan tâm đến khách hàng có cảm nhận về nhà xe như thế nào, dịch vụ có tốt không, nhân viên làm việc tốt không ... Chính vì những vấn đề này đã làm cho khách hàng ít chọn lựa đặt vé xe qua app và làm cho các chủ nhà xe không nắm bắt được tình hình. Vì vậy ứng dụng này sẽ phát triển thêm phần đóng góp ý kiến của khách hàng cho các nhà xe.

1.2. Kế hoạch triển khai.

Tuần	Công việc		Ngày bắt	Ngày kết thúc
3 + 4	Phân tích yêu cầu	Thu thập yêu cầu các chức năng, nghiệp vụ của ứng dụng.	18/9/2023	18/9/2023
		Viết Business Requirement Document	19/9/2023	20/9/2023
		Viết Product Requirement Document	20/9/2023	21/9/2023
	Phân tích và thiết kế	Vẽ cây chức năng, Vẽ use case	22/9/2023	23/9/2023
		Vẽ ERD	24/9/2023	24/9/2023
		Vẽ DFD	25/9/2023	25/9/2023
		Vẽ Activity Diagram	26/9/2023	26/9/2023
		Vẽ Class Diagram	27/9/2023	27/9/2023
	Thiết kế phần mềm	Thiết kế giao diện	28/9/2023	1/10/2023
5 + 6	Viết API và deploy lên server		1/10/2023	15/10/2023
7 → 10	Lập trình		16/10/2023	12/11/2023
11 → 12	Kiểm thử		13/11/2023	26/11/2023

Bảng 1. Kế hoạch triển khai dự án

1.3. Khảo sát tổng quan.

1.3.1. Lịch sử

Trước kia người dùng mua vé xe khách họ phải đến tận nơi để mua, dần dần họ trao đổi với nhau thông qua các cuộc gọi và gửi ảnh vé xe thông qua tin nhắn. Cho đến hiện tại đã có các web/app đặt vé xe khách, nhưng phần lớn những web/app chưa kết nối hết tất cả các nhà xe cả nước. Nên việc đặt vé xe vẫn còn gặp khó khăn, chưa thuận tiện cho khách hàng.

Những vẫn tồn tại một số điểm cho thấy các ứng dụng chưa kết nối tất cả các nhà xe trên khắp tất cả các tỉnh thành: Nhà xe lo ngại rủi ro khi hợp tác với các ứng dụng hiện nay như mất quyền kiểm soát vé, mất khách hàng trung thành, bị cạnh tranh không công bằng.

1.3.2. Các công trình có liên quan

Có một số ứng dụng đặt vé xe hiện nay như là futa, momo, Hai Van, An Phú Quý.

Những app trên cơ bản cũng đã đáp ứng các chức năng đặt vé xe khách như: chọn điểm đi và điểm đến, chọn thời gian khởi hành.

Nhưng những ứng dụng đó không thể đáp ứng nhu cầu đi lại tất cả các tỉnh thành của khách hàng. Như ứng dụng Hai Van chỉ có một vài tuyến đường thuộc khu vực phía Bắc.

Chương 2: Xác định thu thập yêu cầu

2.1. Tổng quan về ứng dụng.

Ứng dụng đặt vé xe khách mang tên “Coach Ticket” với việc bán vé xe khách từ nhiều nhà xe khác nhau. Ứng dụng là cầu nối giữa bộ phận bán vé với khách hàng, giúp khách hàng đặt vé nhanh và thuận tiện hơn và có thể đi khắp tất cả các tỉnh thành trên cả nước. Sau khi đặt vé xe khách hàng có thể kiểm tra lại lịch sử đặt vé.

2.2. Khảo sát hiện trạng nghiệp vụ.

Khách hàng đặt vé trên app Coach Ticket thì phải đăng nhập để tài khoản để có thể truy cập vào app và đặt vé, còn nếu không có tài khoản thì phải đăng ký. Sau khi đăng nhập thành công:

1. Khách hàng tiến hành chọn tuyến đường và ngày xuất phát.
2. Chọn nhà xe, xem thông tin về nhà xe.
3. Chọn ghế ngồi.
4. Thực hiện chọn điểm đón/trả.
5. Điện thoại tin liên hệ.
6. Kiểm tra lại thông tin đặt vé và xác nhận đặt vé.

Sau khi đặt vé có thể xem lịch sử và thực hiện đăng xuất.

2.3. Xác định yêu cầu.

2.3.1. Yêu cầu chức năng.

Ứng dụng đặt vé xe khách có thể thực hiện chức năng của phần mềm như sau:

Tài khoản:

Đăng ký

Đăng nhập

Đăng xuất

Đặt xe:

Tìm kiếm tuyến đường

Hiển thị danh sách thông tin các nhà xe

Chi tiết chuyến xe

Chọn chỗ ngồi

Chọn điểm đón trả

Xác nhận đặt ghế

Hiển thị thông tin đặt vé

Lịch sử đặt xe

Hiển thị danh sách lịch sử đặt vé

Viết bài đánh giá

2.3.2. Yêu cầu phi chức năng.

Giao diện: Giao diện được thiết kế thân thiện với người sử dụng, dễ dàng thao tác đối với khách hàng để đặt vé xe.

Tốc độ xử lý các chức năng phải được thực hiện nhanh chóng.

Tương thích với các thiết bị android.

2.3.3. BRD

2.3.3.1. Tóm tắt dự án.

Hiện tại cần một ứng dụng đặt vé xe tốt và ổn định để đặt vé đi khắp tỉnh đất nước. Các phần đặt vé xe như chọn tuyến đường, thời gian, ghế và xem lịch sử đặt vé thật sự cần phải có. Khách đặt vé thực hiện được thao tác mua hàng trên ứng dụng, để họ có trải nghiệm tốt về ứng dụng sẽ làm cho các nhà xe cảm thấy an tâm với các hoạt động bán vé trực tuyến.

2.3.3.2. Mục tiêu.

Từ những hiện trạng nêu trên, cần thiết kế một phần mềm hỗ trợ cho những nghiệp vụ đặt ra. Với việc dễ dàng hơn khi đặt vé xe ... Tăng tính hiệu quả trong việc đặt vé xe thời gian và công sức đặt vé. Thúc đẩy sự tiện lợi trong vận tải đường bộ.

2.3.3.3. BRD chi tiết.

2.3.3.3.1. Chức năng đăng ký tài khoản.

a. Mục đích và trạng thái hiện tại.

Trạng thái hiện tại: Chức năng đăng ký chưa được số hóa gây khó khăn cho việc mua vé.

Qua quá trình thu thập yêu cầu và khảo sát nhận thấy hệ thống cần phải có chức năng đăng ký tài khoản dành cho khách hàng, tạo điều kiện thuận lợi cho việc mua vé.

b. Yêu cầu nghiệp vụ

Thực hiện đăng ký tài khoản cho khách hàng từ những thông tin người dùng cung cấp và bảo mật thông tin cho khách hàng, lưu trữ dữ liệu trên server (cyclic).

Nếu đăng ký thành công sẽ xuất hiện thông báo đăng ký thành công và hiển thị lên màn hình của ứng dụng.

Nếu đăng ký không thành công sẽ hiển thị lỗi cụ thể.

c. Yêu cầu logic

Các trường nhập vào không được để trống

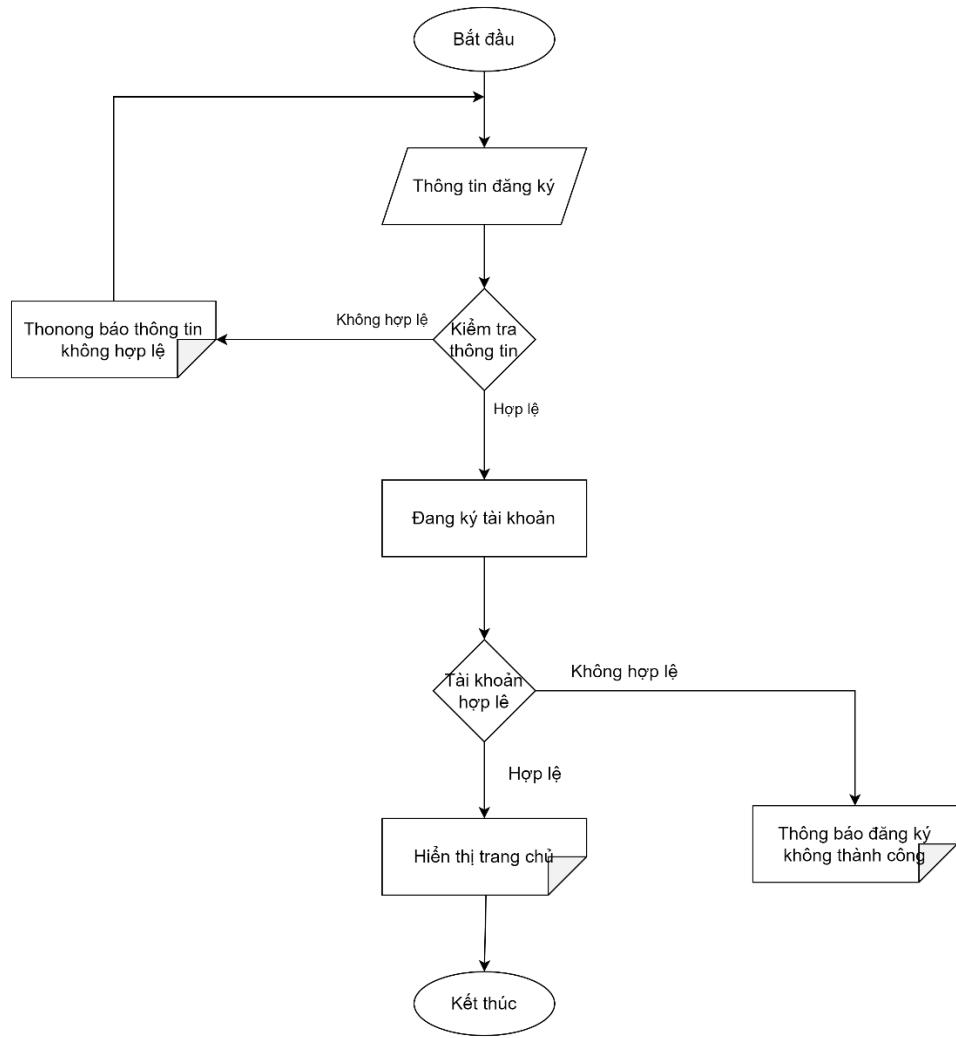
Username phải có định dạng email, không được chứa khoảng trắng

Địa chỉ email mới chưa được đăng ký trước đây

Password phải có độ dài ít nhất 6 ký tự và không được chứa khoảng trắng

Password phải được ẩn khi người dùng nhập

d. Workflow



Hình 1. Workflow chức năng đăng ký.

Mô tả workflow:

Bước 1: Bắt đầu

Bước 2: Nhận thông tin đăng ký do người dùng nhập vào

Bước 3: Kiểm tra tính hợp lệ của các thông tin bao gồm các yếu tố :

- Các trường nhập vào không được để trống
- Username phải có định dạng email, không được chứa khoảng trắng
- Password phải có độ dài ít nhất 8 ký tự và không được chứa khoảng trắng

Bước 4: Nếu thông tin hợp lệ thì bỏ qua bước 5

Bước 5: Thông báo thông tin nhập vào không hợp lệ và quay về bước 2

Bước 6: Thực hiện đăng ký tài khoản trên cơ sở dữ liệu

Bước 7: Nếu đăng ký thành công thì bỏ qua bước 8

Bước 8: Hiển thị lỗi đăng ký không thành công

Bước 9: Hiển thị trang chủ hệ thống

Bước 10: Kết thúc

2.3.3.3.2. BRD chức năng đăng nhập

a. Mục đích

Với mỗi khách hàng, cần phải có một tài khoản riêng để lưu trữ thông tin về khách hàng, thực hiện mua vé và thanh toán, cũng như lịch sử mua vé,... cho phép ứng dụng lưu trữ dữ liệu người dùng một cách an toàn và cung cấp trải nghiệm được cá nhân hóa giống nhau trên tất cả các thiết bị của người dùng và có thể giúp cho thông tin cá nhân của những người sử dụng được an toàn hơn. Điều này cũng đảm bảo tài khoản và các thông tin cá nhân của người dùng không bị đánh cắp.

Từ những vấn đề trên, hệ thống bắt buộc phải có chức năng đăng nhập.

b. Yêu cầu nghiệp vụ

Thực hiện đăng nhập vào hệ thống với username và password do người dùng nhập vào thông qua giao diện đăng nhập.

Nếu đăng nhập thành công sẽ hiển thị trang chủ của ứng dụng tùy theo tài khoản của người dùng

Nếu đăng nhập không thành công, hệ thống sẽ hiển thị thông báo lỗi cho người dùng

c. Yêu cầu logic

Các trường nhập vào không được để trống

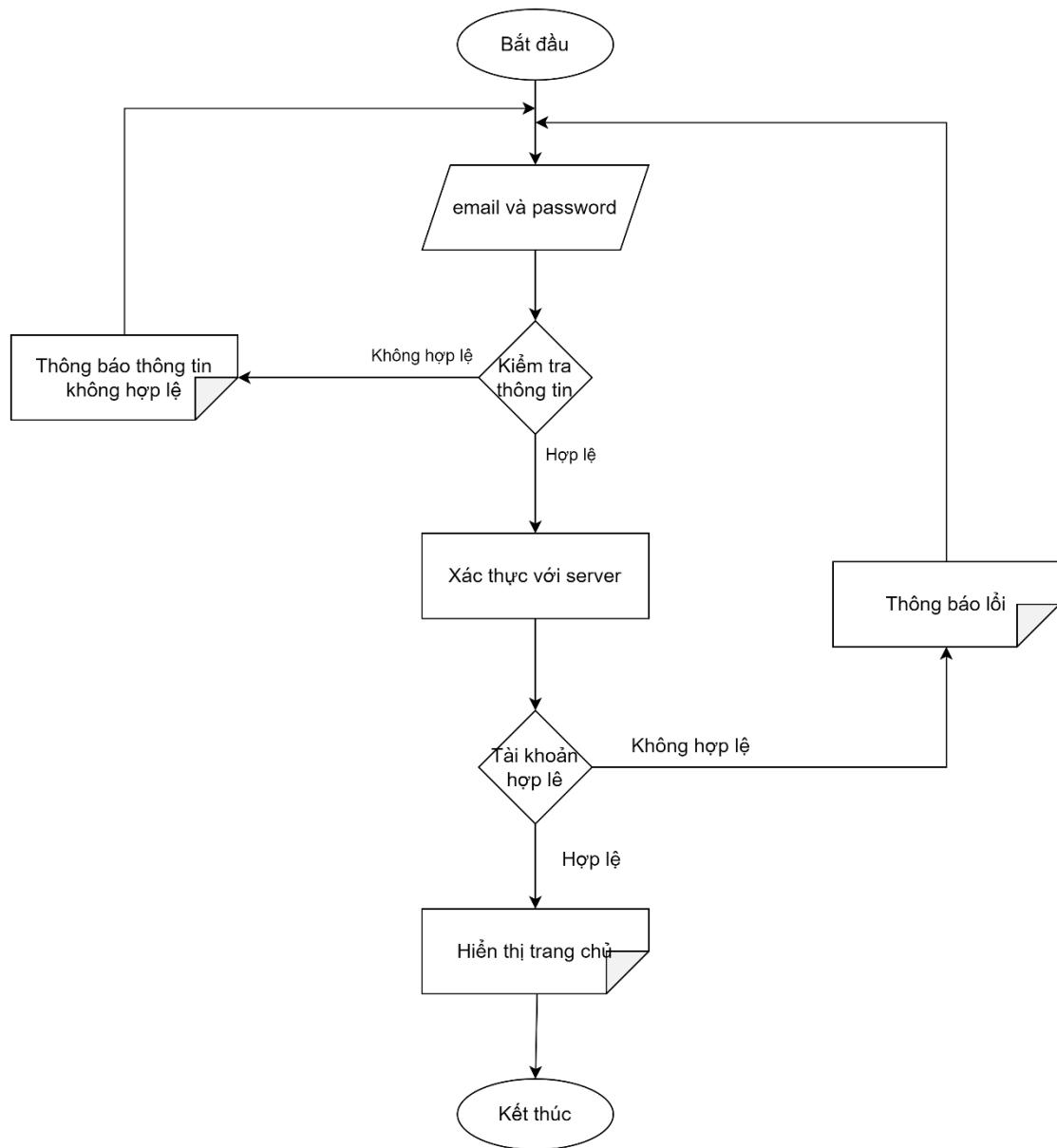
Username phải có định dạng email, không được chứa khoảng trắng

Địa chỉ email mới chưa được đăng ký trước đây

Password phải có độ dài ít nhất 8 ký tự và không được chứa khoảng trắng

Password phải được ẩn khi người dùng nhập

d. Workflow



Hình 2. Workflow chức năng đăng nhập.

Mô tả workflow:

Bước 1: Bắt đầu

Bước 2: Username và password do người dùng nhập vào.

Bước 3: Kiểm tra tính hợp lệ của username và password bao gồm các yếu tố :

Username và password không được để trống

Username phải có định dạng email, không được chứa khoảng trắng

Password phải có độ dài ít nhất 6 ký tự và không được chứa khoảng trắng

Bước 4: Nếu Username và password hợp lệ thì bỏ qua bước 5

Bước 5: Thông báo lỗi username và password không hợp lệ ở màn hình đăng nhập và quay về bước 2

Bước 6: Thực hiện xác thực firebase với Username và password từ bước 4, nếu hợp lệ thì bỏ qua bước 7

Bước 7: Thông báo lỗi tài khoản đăng nhập không thành công ở màn hình đăng nhập và quay về bước 2

Bước 8: Hiển thị giao diện màn hình chính của ứng dụng

Bước 9: Kết thúc

2.3.3.3. BRD chức năng đặt vé xe

a. Mục đích và trạng thái hiện tại

Trạng thái hiện tại: Hiện tại còn nhiều khách hàng vẫn còn mua vé bằng các hình thức cũ thông qua cuộc gọi và vé giấy.

Mục đích: Nghiên cứu về các hoạt động mua vé trực tuyến của khách hàng, giúp khách hàng dễ dàng đặt vé xe.

Từ những khảo sát thực tế và thu thập yêu cầu, nhận thấy chức năng đặt vé cần đáp ứng đầy đủ chức năng để phục vụ quá trình đặt vé của khách hàng được nhanh chóng và dễ dàng: Chọn tuyến đường, hiển thị danh sách thông tin các nhà xe, chi tiết thông tin nhà xe, chọn chỗ ngồi, chọn điểm đón/trả, xác nhận đặt ghế.

b. Yêu cầu nghiệp vụ

Về thao tác

Người dùng cần đăng nhập tài khoản khách hàng hoặc tạo một tài khoản mới nếu chưa có tài khoản trước khi đặt vé.

Về chức năng:

Về chức năng chọn tuyến đường: Khách hàng có thể nhập tĩnh thành và ngày.

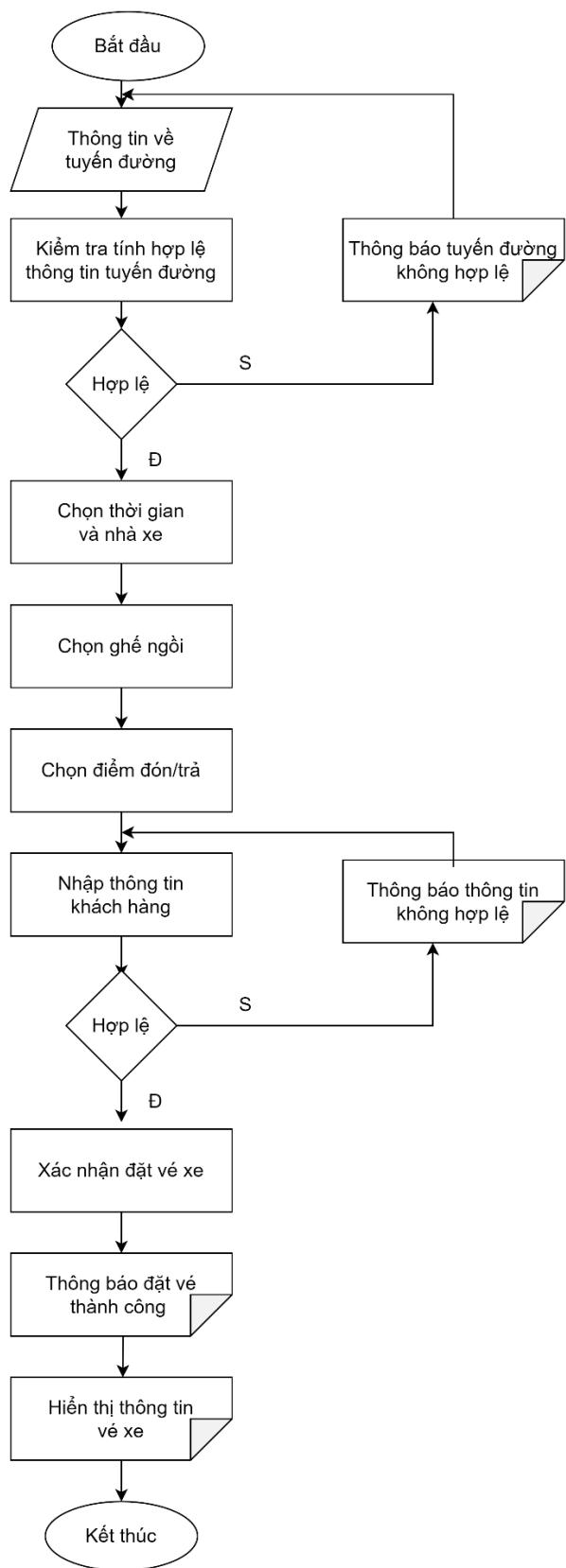
Về chức năng chọn chỗ ngồi: khách hàng có thể chọn số ghế được hiển thị trên màn hình.

Về chọn điểm đón trả: Khách hàng có thể chọn điểm đón trả theo danh sách.

c. Yêu cầu logic

Các thông tin nhập tuyển đường không được để trống.

d. Workflow



Hình 3. Workflow chức năng đặt vé xe.

Mô tả workflow đặt vé xe:

Bước 1: Bắt đầu

Bước 2: Nhập thông tin về tuyến đường, điểm đi và điểm đến

Bước 3: Kiểm tra tính hợp lệ của tuyến đường

Bước 4: Nếu thông tin hợp lệ thì bỏ qua bước 5

Bước 5: Thông báo thông tin nhập vào không hợp lệ và quay về bước 2

Bước 6: Chọn thời gian

Bước 7: Chọn ghế

Bước 8: Chọn điểm đón/trả

Bước 9: Nhập thông tin khách hàng

Bước 10: Nếu thông tin đúng thì bỏ qua bước 11

Bước 11: Nếu thông tin không hợp lệ thì quay lại bước 9

Bước 12: Xác nhận đặt vé

Bước 13: Thông báo đặt vé thành công

Bước 14: Hiển thị thông tin vé xe

2.3.4.4. BRD chức năng xem lịch sử đặt vé.

a. Mục đích và trạng thái hiện tại

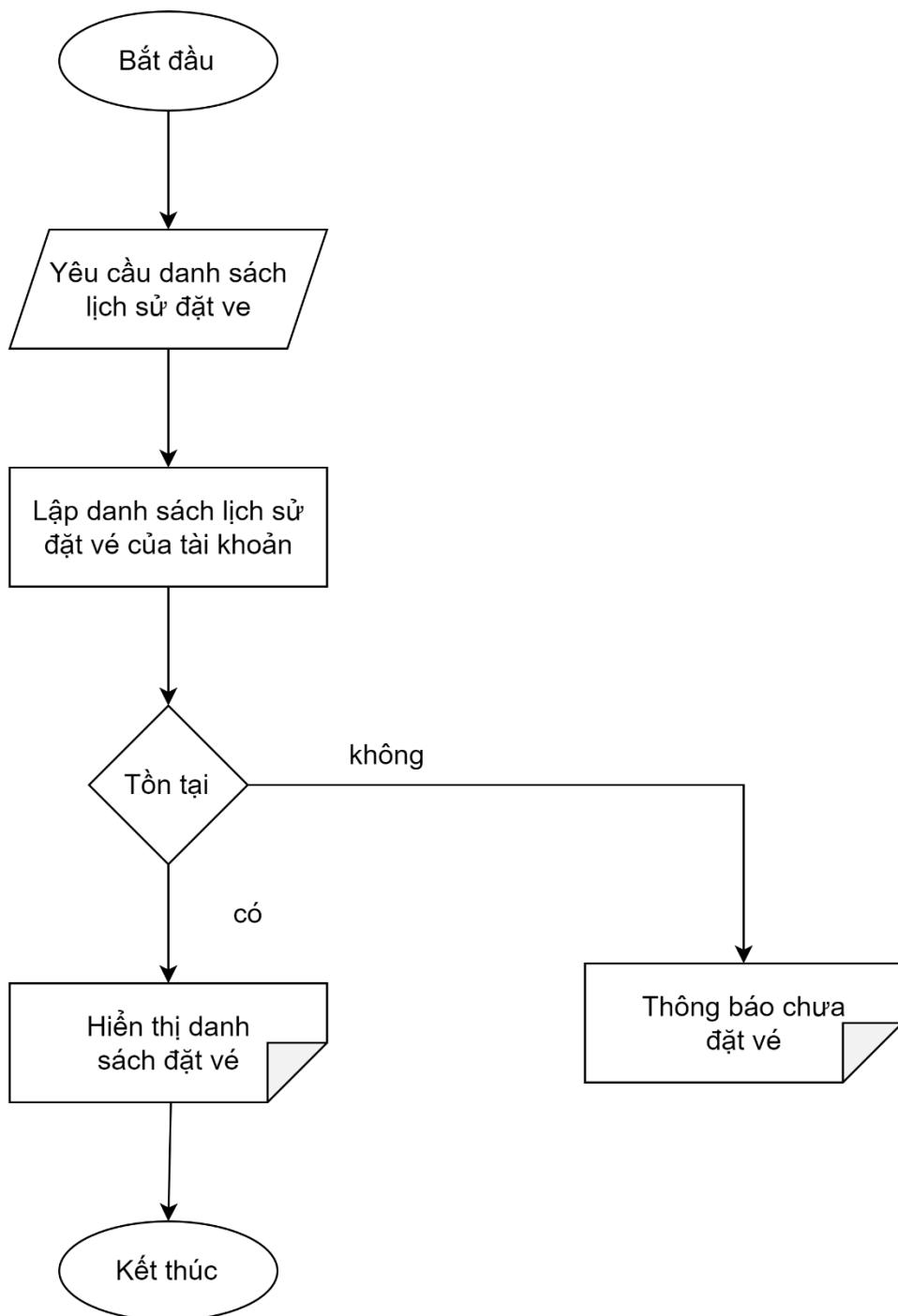
Trạng thái hiện tại: Trong quá trình khảo sát và thu thập yêu cầu, nhận thấy cần phải có chức năng quản lý và theo dõi các vé đã đặt.

b. Yêu cầu nghiệp vụ

Về thao tác

Người dùng cần đăng nhập tài khoản khách hàng hoặc tạo tài khoản mới nếu chưa có tài khoản trước khi thực hiện theo dõi lịch sử đặt vé.

c. workflow



Hình 4. Workflow chức năng xem lịch sử đặt vé.

Mô tả workflow

Bước 1: Bắt đầu

Bước 2: Yêu cầu thông tin lịch sử đặt vé từ khách hàng

Bước 3: Lấy danh sách khách hàng từ cơ sở dữ liệu

Bước 4: Nếu có lịch sử đặt vé thì bỏ qua bước 5

Bước 5: Thông báo không có lịch sử đặt vé

Bước 6: Hiển thị danh sách lịch sử đặt vé của tài khoản.

Bước 7: Kết thúc

2.3.4. PRD

2.3.4.1. PRD chức năng đăng ký.

a. Nghịệp vụ.

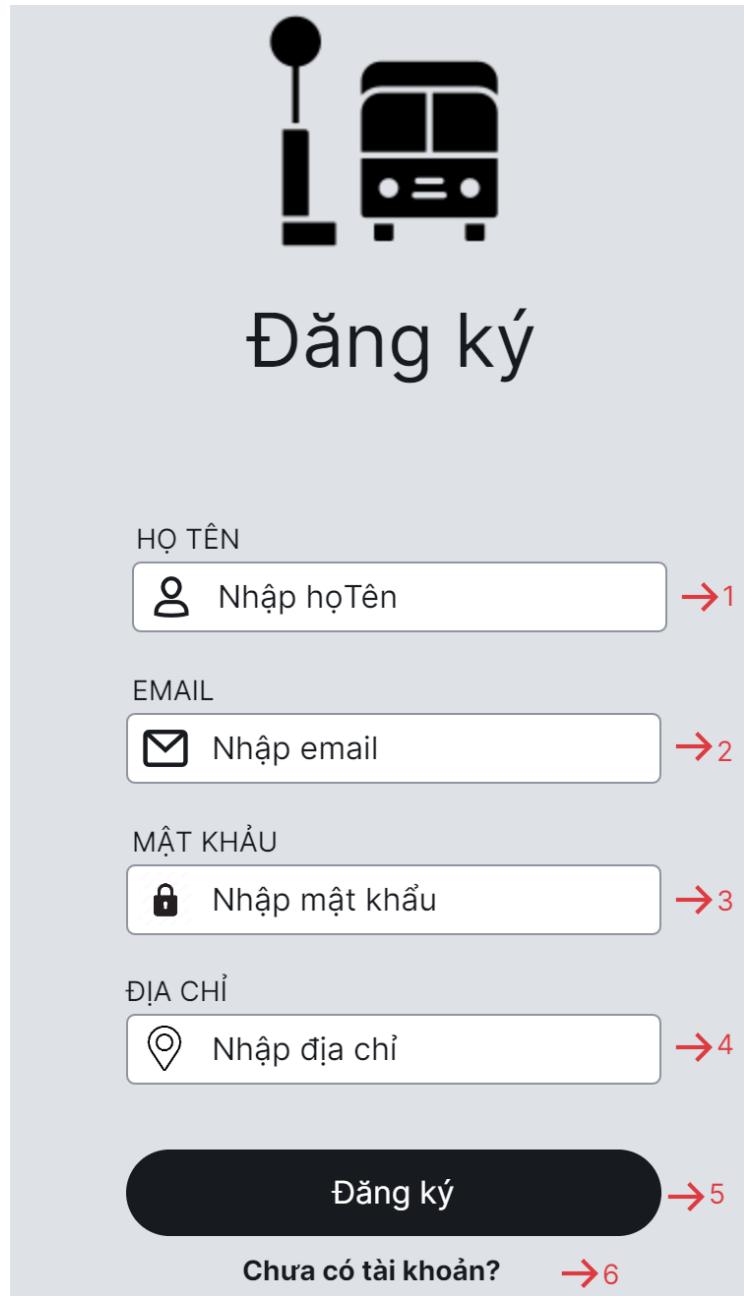
Từ giao diện màn hình đăng nhập, Nếu người dùng chưa có tài khoản thì có thể chọn nút đăng ký tài khoản mới để chuyển sang giao diện đăng ký và thực hiện đăng ký tài khoản mới.

Sau khi người dùng nhập đầy đủ các trường thông tin bao gồm tên người dùng, Email, mật khẩu, địa chỉ. Hệ thống thực hiện kiểm tra tính hợp lệ của các thông tin thỏa mãn các ràng buộc :

- Các trường không được bỏ trống
- Email phải có định dạng email, không được chứa khoảng trắng
- Mật khẩu phải có độ dài ít nhất 8 ký tự và không được chứa khoảng trắng

Nếu không hợp lệ sẽ xuất hiện thông báo thông tin không hợp lệ, nếu hợp lệ, hệ thống thực hiện đăng ký tài khoản mới và ghi dữ liệu vào cơ sở dữ liệu. Nếu đăng ký thành công sẽ hiện thông báo thành công và chuyển người dùng vào trang chủ của hệ thống, nếu không sẽ hiển thị thông báo lỗi.

b. UI mockup.



Hình 5. UI mockup đăng ký.

c. Phân tích UI mockup.

Số 1: Trường để nhập Tên khách hàng

Số 2: Trường để nhập Email

Số 3: Trường để nhập mật khẩu

Số 4: Trường để nhập Địa chỉ

Số 5: Nút Đăng ký, Khi nhấn sẽ thực hiện quá trình đăng ký

Số 6: Text đăng nhập, Khi nhấn sẽ chuyển sang chức năng đăng nhập

2.3.4.2. PRD chức năng đăng nhập.

a. Nghiệp vụ.

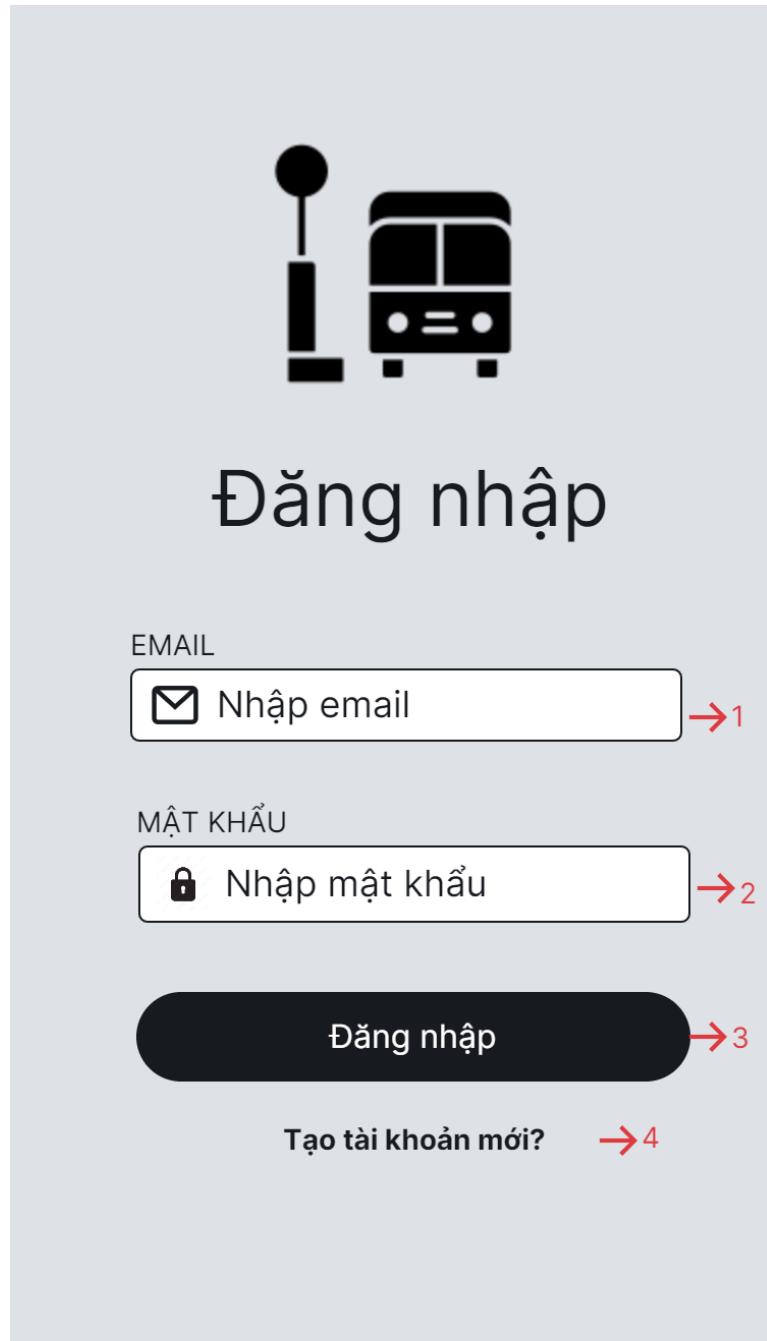
Tùy giao diện màn hình đăng nhập, sau khi người dùng nhập email và mật khẩu vào đầy đủ các trường, người dùng ấn nút đăng nhập để thực hiện quá trình đăng nhập

Hệ thống thực hiện kiểm tra tính hợp lệ của các trường email và mật khẩu thỏa mãn các ràng buộc :

- Email và Mật khẩu không được để trống
- Email phải có định dạng email, không được chứa khoảng trắng
- Mật khẩu phải có độ dài ít nhất 8 ký tự và không được chứa khoảng trắng

Nếu không hợp lệ sẽ xuất hiện thông báo email hoặc mật khẩu không hợp lệ, nếu hợp lệ hệ thống thực hiện xác thực để kiểm tra tính hợp lệ của tài khoản nếu thành công thực hiện chuyển sang giao diện trang chủ. nếu có vấn đề hoặc lỗi sẽ xuất hiện thông báo lỗi cho người dùng.

b. UI mockup.



Hình 6. UI mockup đăng nhập.

c. Phân tích UI mockup

Số 1: Trường để nhập email

Số 2: Trường để nhập mật khẩu

Số 3: Nút đăng nhập, Khi nhấn sẽ thực hiện quá trình đăng nhập

Số 4: Text đăng ký, Khi nhấn sẽ chuyển sang chức năng đăng ký tài khoản mới

2.3.4.3. PRD chức năng đặt vé xe.

2.4.3.3.1. PRD chức năng chọn tuyến đường.

a. Nghịệp vụ.

Sau khi người dùng chọn đầy đủ các trường thông tin gồm điểm đi, điểm đến và ngày đi. Hệ thống sẽ thực hiện kiểm tra tính hợp lệ của các trường thông tin thỏa mãn ràng buộc các trường không được bỏ trống.

Nếu không hợp lệ sẽ xuất hiện thông báo thông tin không hợp lệ, nếu hợp lệ hệ thống sẽ ghi dữ liệu và chuyển đến phần tiếp theo.

b. UI mockup



Hình 7. UI mockup chọn tuyến đường.

c. Phân tích UI mockup:

Số 1: Chọn điểm đi

Số 2: Chọn điểm đến

Số 3: Chọn ngày đi

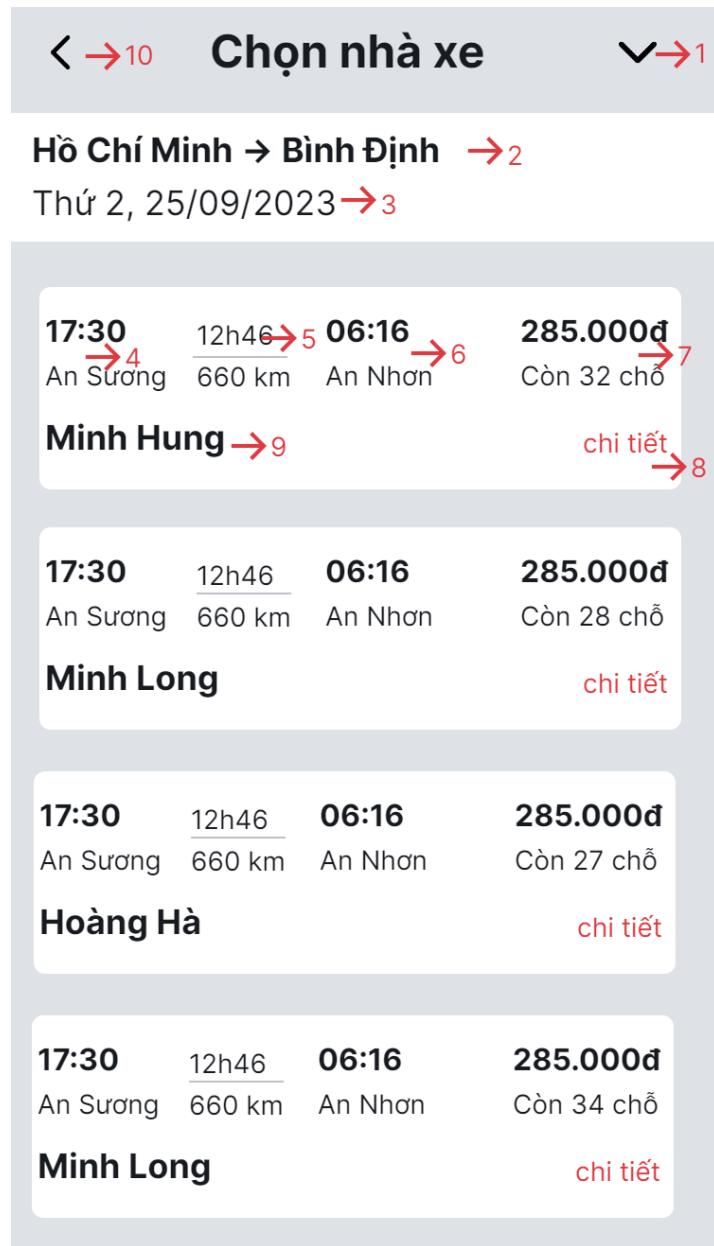
Số 4: Nút tìm kiếm, Khi nhấn chuyển sang mà hình tiếp theo

2.4.3.3.2. PRD chức năng chọn nhà xe.

a. Nghề vụ.

Với chức năng chọn nhà xe: Các nhà xe sẽ được hiển thị trên màn hình, có thời gian địa điểm và tên của từng nhà xe. Khách hàng có thể chọn bất kì nhà xe nào cảm thấy phù hợp.

b. UI mockup



Hình 8. UI mockup chọn nhà xe.

c. Phân tích UI mockup

Số 1: Là icon lọc các nhà xe.

Số 2: Hiển thị tuyến đường đi và điểm đến

Số 3: Hiển thị ngày đi

Số 4: Hiển thị thời gian và địa điểm bến xe của điểm đi

Số 5: Hiển thị khoảng thời gian đi và khoảng cách tuyền đường

Số 6: Hiển thị thời gian và địa điểm bến xe điểm đến

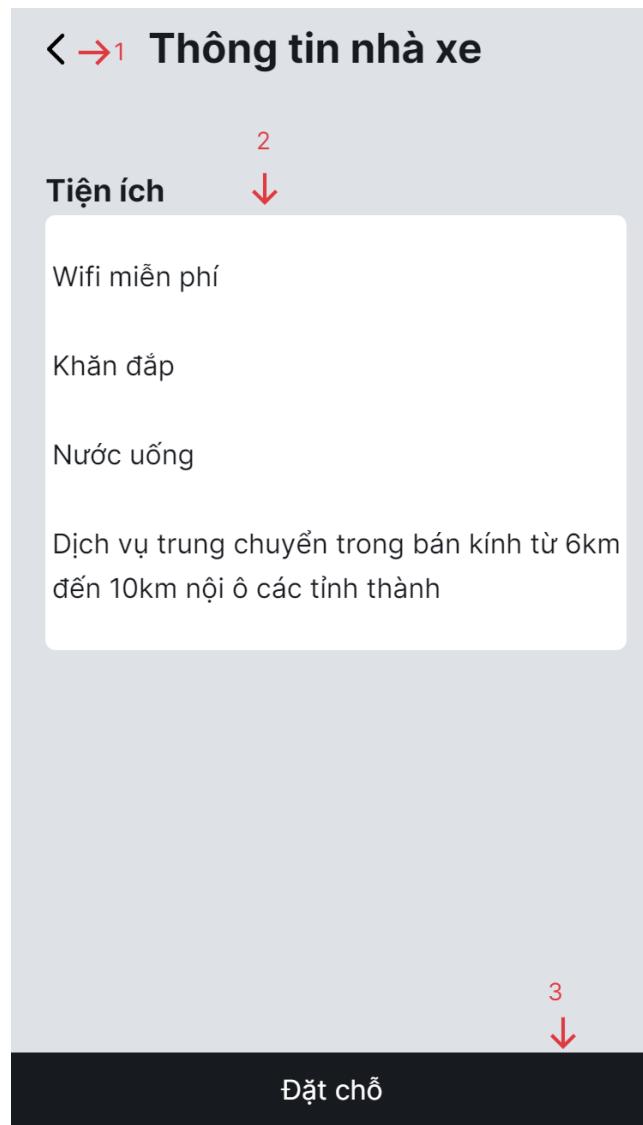
Số 7: Hiển thị giá tiền và chỗ ngồi còn trống

Số 8: Hiển thị thông tin chi tiết nhà xe

Số 9: Hiển thị tên nhà xe

Số 10: Nút quay trở lại màn hình trước

d. UI mockup thông tin nhà xe



Hình 9. UI mockup thông tin nhà xe.

e. Phân tích UI mockup thông tin nhà xe.

Số 1: Nút quay lại màn hình trước

Số 2: Hiển thị thông tin tiện ích nà xe

Số 3: Nút đặt chỗ, sau khi khách hàng nhấn sẽ chuyển sang màn hình đặt chỗ

2.4.3.3. PRD chức năng chọn chỗ ngồi.

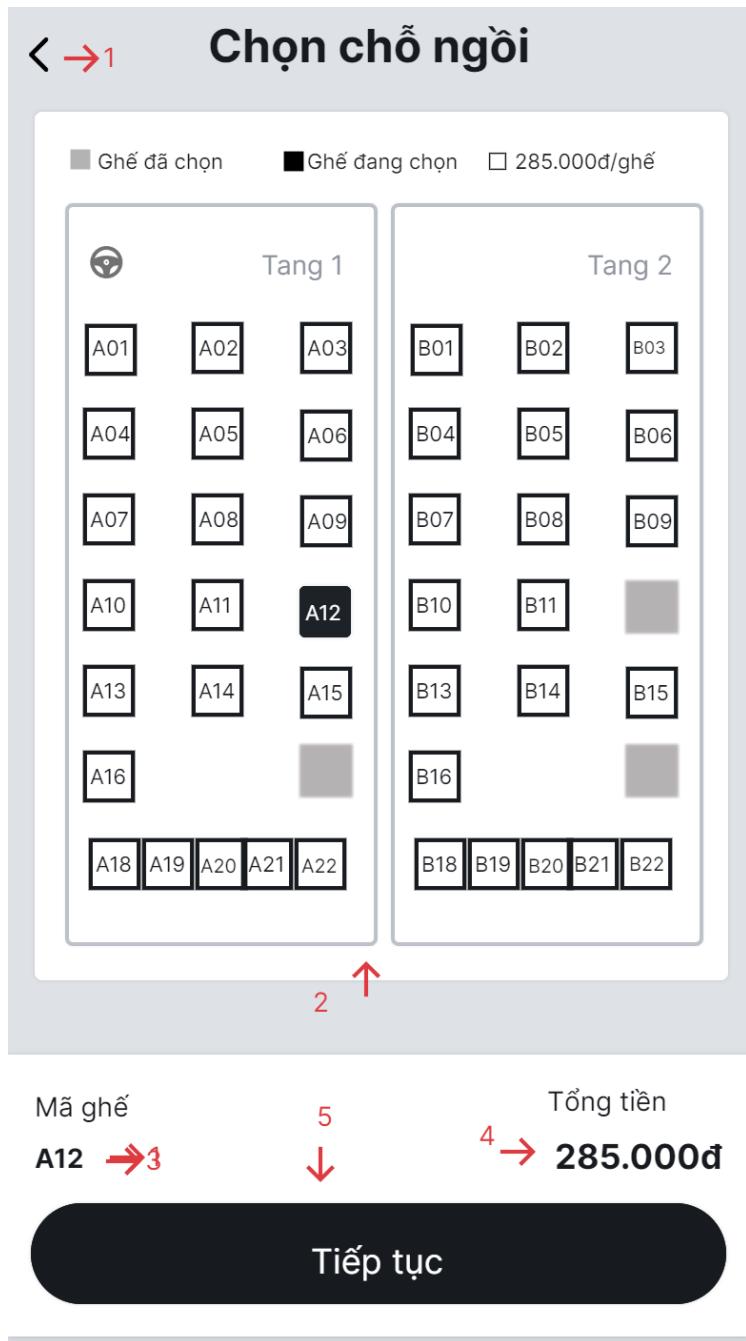
a. Nghịệp vụ.

Khách hàng chọn ghế ngồi được hiển thị trên màn hình, sau khi chọn ghế thì thông tin mã ghế sẽ hiển thị trên màn hình và tính tổng số tiền. Khách hàng có thể chọn nhiều ghế ngồi.

Khách hàng có thể bỏ chọn nếu thấy không ưng ý. Sau khi chọn xong nhấn nút tiếp theo để chuyển sang màn hình tiếp theo.

Hệ thống sẽ kiểm tra người dùng đã chọn ghế chưa, nếu chưa chọn hệ thống sẽ thông báo, nếu đã chọn hệ thống sẽ chuyển sang màn hình tiếp theo

b. UI mockup



Hình 10. UI mockup chọn ghế ngồi.

c. Phân tích UI mockup

Số 1: Nút quay lại màn hình trước

Số 2: Các hàng ghế để khách hàng lựa chọn

Số 3: Hiển thị mã số ghế sau khi khách hàng chọn

Số 4: Hiển thị tổng số tiền ghé

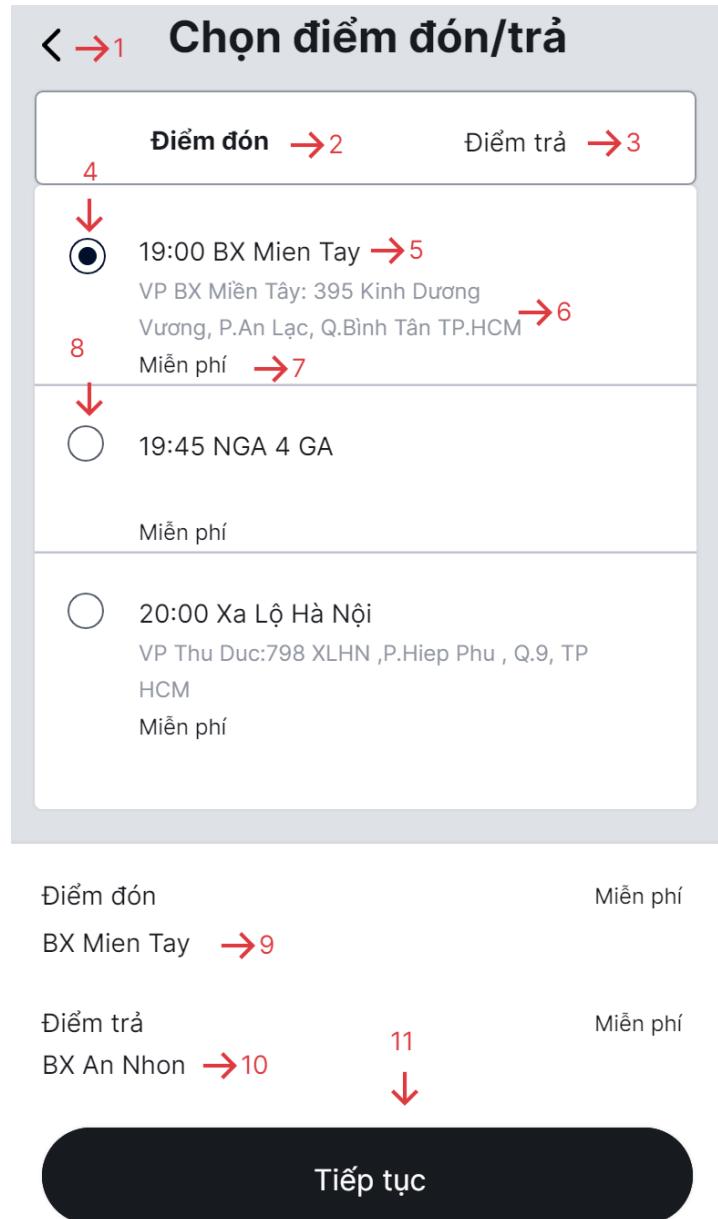
Số 5: Nút tiếp tục, sau khi nhấn sẽ chuyển sang màn hình khác.

2.4.3.3.4. PRD chức năng chọn điểm đón/trả.

a. Nghịệp vụ.

Khách hàng chọn điểm đón và điểm trả, sau khi chọn xong khách hàng chọn nút tiếp theo để chuyển sang màn hình mới. Nếu chưa chọn điểm trả và điểm đón hệ thống sẽ thông báo lỗi, nếu đã chọn hệ thống sẽ chuyển sang màn hình mới.

b. UI mockup



Hình 11. UI mockup chọn điểm đón, điểm trả.

c. Phân tích UI mockup

Số 1: Nút quay lại màn hình trước

Số 2: Text chọn để hiển thị danh sách điểm đón

Số 3: Text chọn để hiển thị danh sách điểm trả

Số 4: radio cho biết khách hàng đã chọn điểm đón

Số 5: Thời gian và điểm đón

Số 6: Địa điểm chi tiết

Số 7: Hiển thị thông tin cho biết điểm đón miễn phí

Số 8: radio cho biết khách hàng không chọn địa điểm đó

Số 9: Sau khi chọn điểm đón, text sẽ hiển thị thông tin

Số 10: Sau khi chọn điểm trả hệ thống sẽ hiển thị thông tin

Số 11: Nút tiếp tục, sau khi khách hàng chọn sẽ chuyển sang màn hình tiếp theo

2.4.3.3.5. PRD chức năng nhập thông tin liên hệ.

a. Nghiệp vụ.

Khách hàng nhập thông tin theo yêu cầu của ứng dụng họ tên, email, số điện thoại. Hệ thống thực hiện kiểm tra tính hợp lệ của các thông tin thỏa mãn các ràng buộc:

- Các trường không được bỏ trống
- Email phải đúng định dạng, không được chứa khoảng trắng
- Số điện thoại phải đảm bảo đủ mười số.

Nếu không hợp lệ sẽ xuất hiện thông báo thông tin không hợp lệ, nếu hợp lệ hệ thống sẽ chuyển sang màn hình mới.

b. UI mockup

< →⁰ Thông tin liên hệ

Họ và tên
Nguyễn Văn An →¹

email
nguyenvanan@gmail.com →²

Số điện thoại
0347398738 →³

4
↓

Đặt Chỗ

Hình 12. UI mockup thông tin liên hệ.

c. Phân tích UI mockup

Số 0: Nút quay lại màn hình trước

Số 1: Trường để nhập họ tên

Số 2: Trường để nhập email

Số 3: Trường để nhập số điện thoại

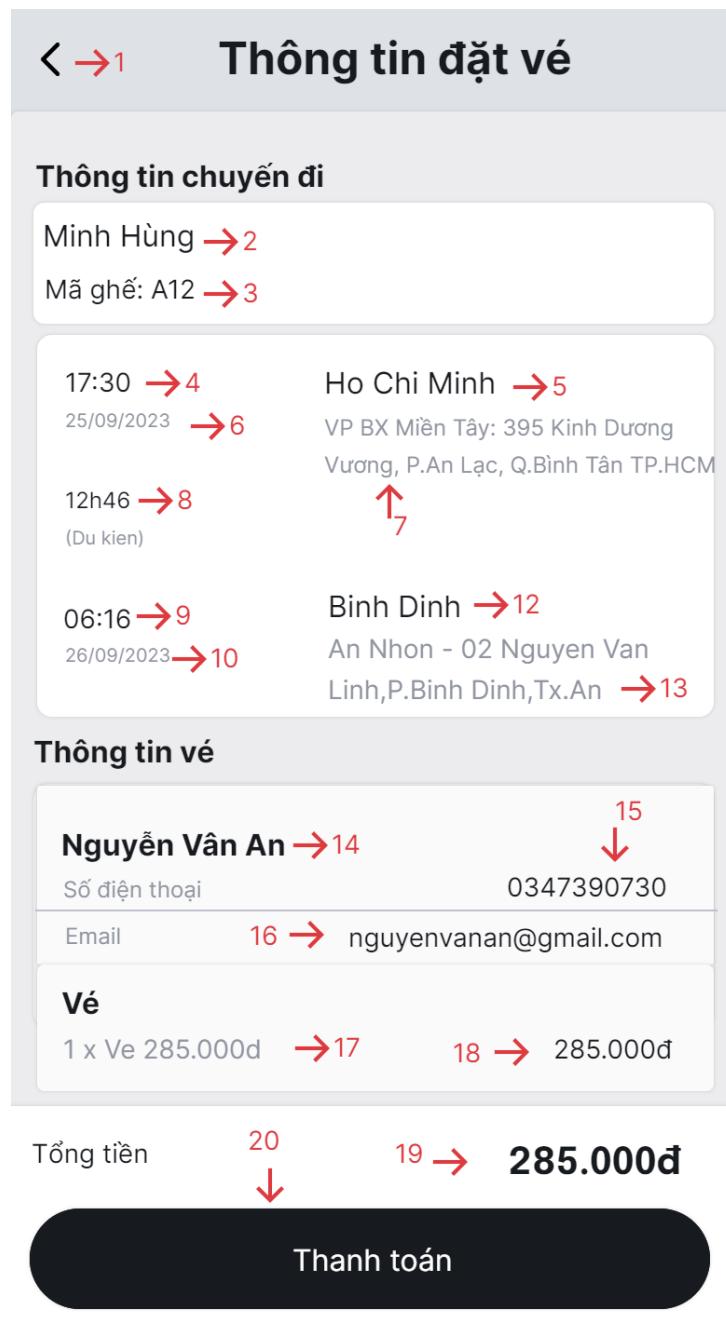
Số 4: Nút đặt chỗ, khi nhấn nút sẽ chuyển sang màn hình tiếp theo

2.4.3.3.6. PRD hiển thị thông tin đặt vé.

a. Nghịệp vụ

Sau khi khách hàng xem và xác nhận thông tin đặt vé xe về thông tin chuyến xe, thông tin vé và số tiền đúng thì nhấn nút thanh toán để chuyển sang màn hình tiếp theo.

b. UI mockup



Hình 13. UI mockup thông tin đặt vé.

c. Phân tích UI mockup.

Số 1: Nút quay lại màn hình trước

Số 2: Hiển thị thông tin nhà xe

Số 3: Hiển thị mã ghế đã đặt

Số 4: Hiển thị thời gian điểm đi

Số 5: Hiển thị điểm đi

Số 6: Hiển thị ngày đi

Số 7: Hiển thị chi tiết điểm đón.

Số 8: Khoảng thời gian dự kiến của chuyến đi

Số 9: Hiển thị thời gian điểm đến

Số 10: Hiển thị ngày đến

Số 12: Hiển thị điểm đến

Số 13: Hiển thị thông tin chi tiết điểm trả

Số 14: Hiển thị tên khách hàng

Số 15: Hiển thị số điện thoại khách hàng

Số 16: Hiển thị email khách hàng

Số 17: Hiển thị số vé và giá tiền cho một vé.

Số 19: Hiển thị tổng số tiền

Số 20: Nút Thanh toán, sau khi khách hàng nhấn nút sẽ chuyển sang màn hình mới.

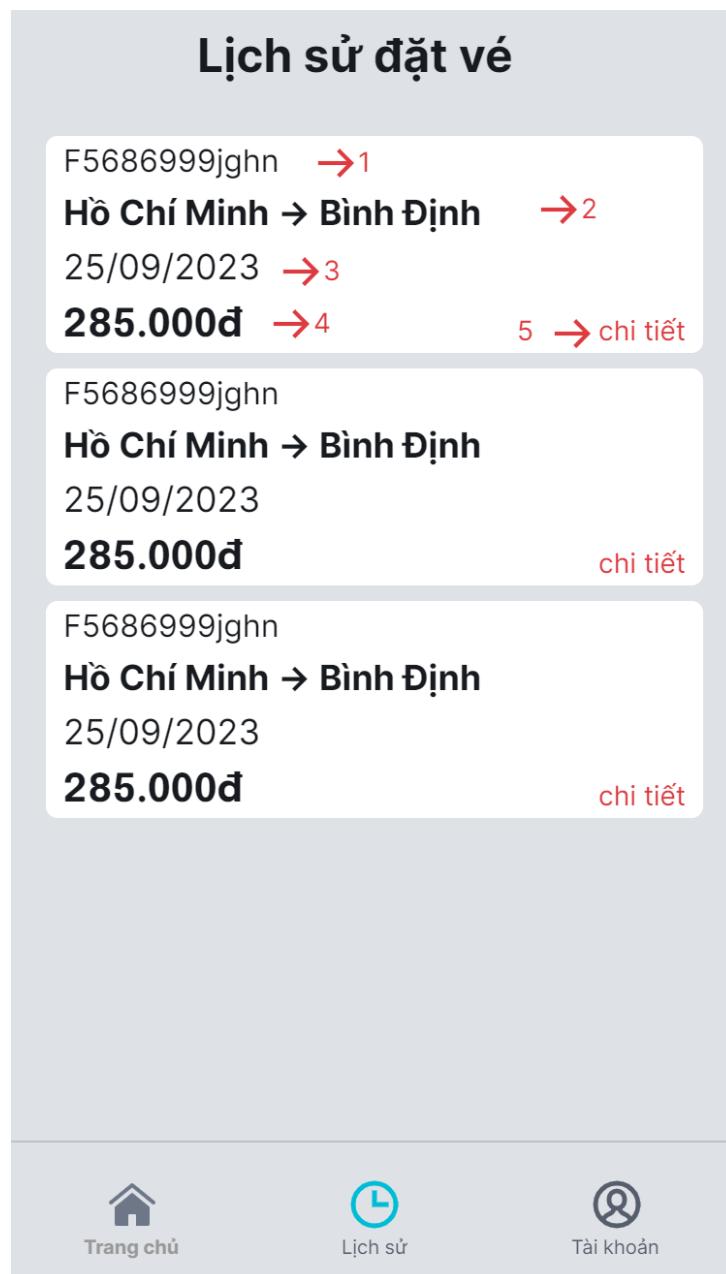
2.3.4.7. PRD chức năng xem lịch sử đặt vé xe.

2.3.4.7.1. PRD danh sách lịch sử vé xe.

a. Nghịệp vụ.

Hiển thị danh sách vé xe mà khách hàng đã đặt mua, hiển thị các thông tin mã vé, tuyến đường, ngày và giá vé.

b. UI mockup



Hình 14. UI mockup lịch sử đặt vé.

c. Phân tích UI mockup

Số 1: hiển thị mã vé

Số 2: Hiển thị tuyến đường

Số 3: Hiển thị ngày đi

Số 4: Hiển thị tổng giá tiền

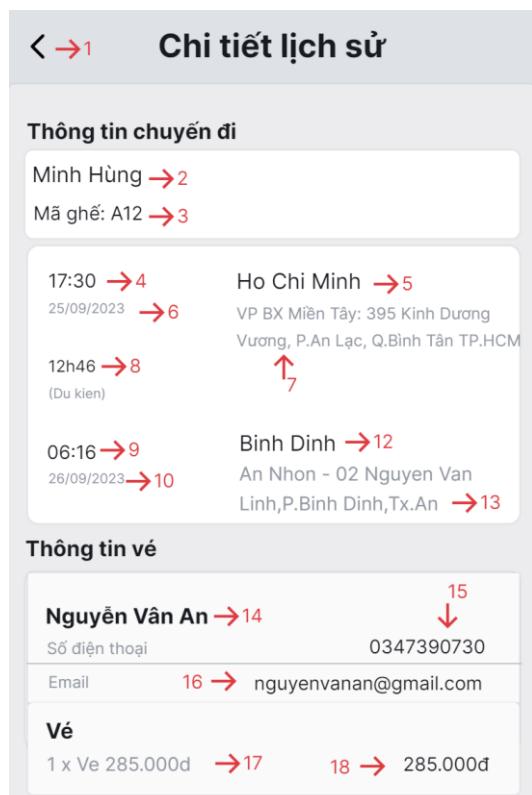
Số 5: Text xem chi tiết, sau khi khách hàng nhấn vào sẽ hiển thị chi tiết vé

2.3.4.7.1. PRD danh sách lịch sử vé xe.

a. Nghịệp vụ.

Khách hàng có thể xem lại chi tiết lịch sử đặt vé và cũng có thể dùng để lên xe.

b. UI mockup.



Hình 15. UI mockup chi tiết lịch sử đặt vé.

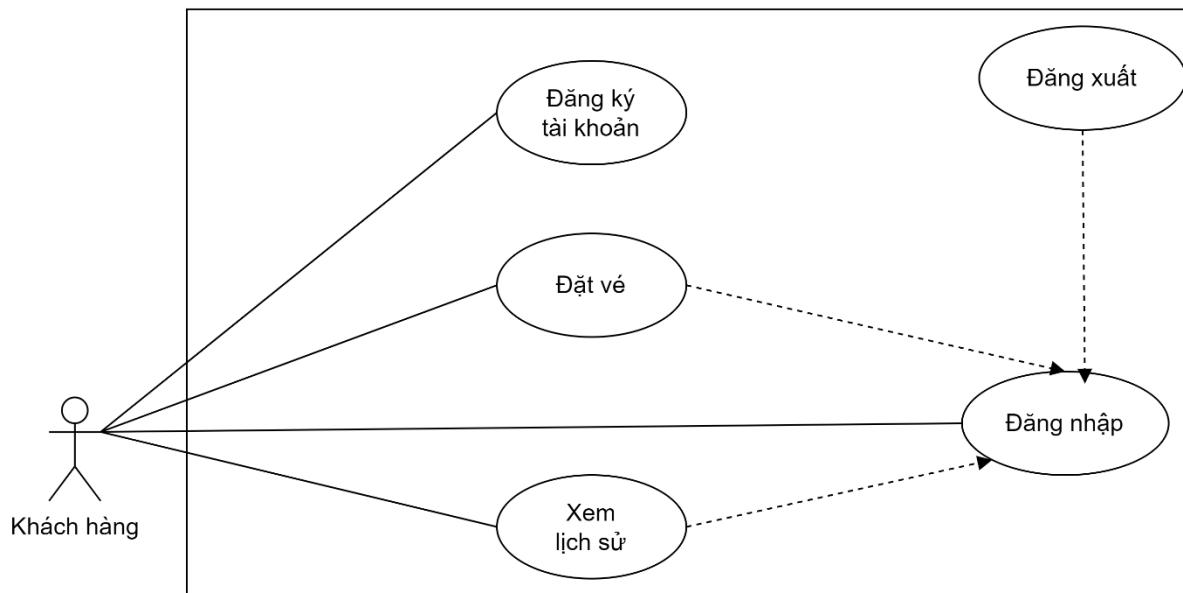
C. Phân tích UI mockup.

- Số 1: Nút quay lại màn hình trước
- Số 2: Hiển thị thông tin nhà xe
- Số 3: Hiển thị mã ghế đã đặt
- Số 4: Hiển thị thời gian điểm đi
- Số 5: Hiển thị điểm đi
- Số 6: Hiển thị ngày đi
- Số 7: Hiển thị chi tiết điểm đón.
- Số 8: Khoảng thời gian dự kiến của chuyến đi
- Số 9: Hiển thị thời gian điểm đến
- Số 10: Hiển thị ngày đến
- Số 12: Hiển thị điểm đến
- Số 13: Hiển thị thông tin chi tiết điểm trả
- Số 14: Hiển thị tên khách hàng
- Số 15: Hiển thị số điện thoại khách hàng
- Số 16: Hiển thị email khách hàng
- Số 17: Hiển thị số vé và giá tiền cho một vé.
- Số 18: Hiển thị tổng số tiền

Chương 3: Phân tích thiết kế

3.1. Use case.

3.1.1. Use case tổng quát.



Hình 16. Use case tổng quát.

Mô tả Actor:

Tên Actor	Mô tả
Khách hàng	Truy cập vào ứng dụng để đặt mua vé xe, xem lịch sử đặt ve.

Bảng 2. Mô tả actor

Mô tả các use case:

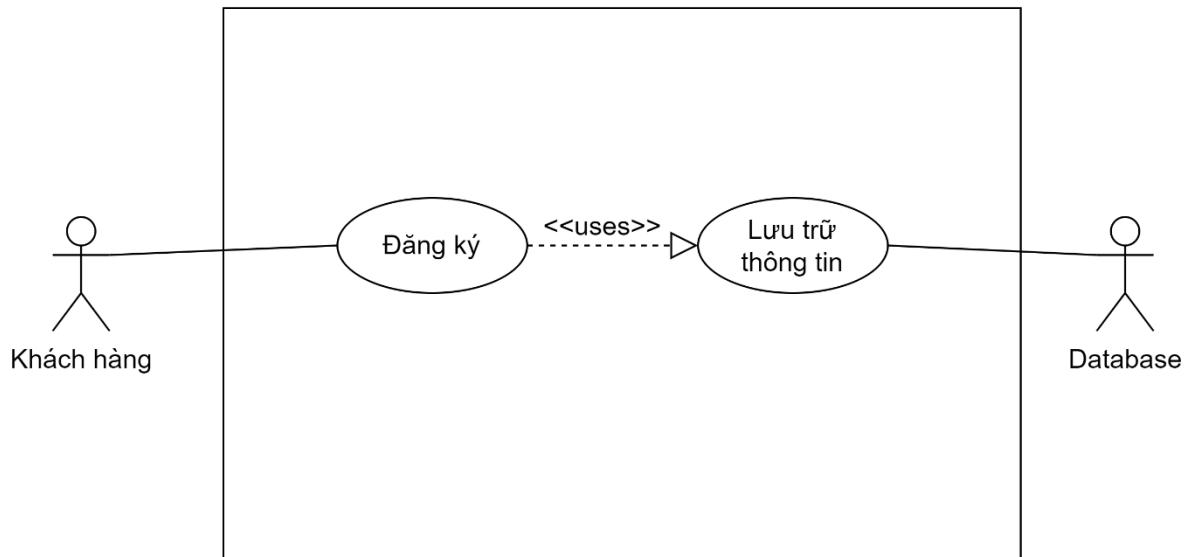
Số thứ tự	Tên Use case	Mô tả
1	Đăng ký	Cho phép khách hàng đăng ký và sử dụng hệ thống
2	Đăng nhập	Cho phép khách hàng đăng nhập vào hệ thống
3	Đăng xuất	Cho phép khách hàng đăng xuất khỏi hệ thống
4	Đặt vé	Cho phép khách hàng thực hiện đặt vé

5	Xem lịch sử	Cho phép khách hàng xem lịch sử đặt vé
---	-------------	--

Bảng 3. Mô tả các use case

3.1.2. Phân rã và đặc tả use case.

3.1.2.1. Use case đăng ký.



Hình 17. Use case đăng ký.

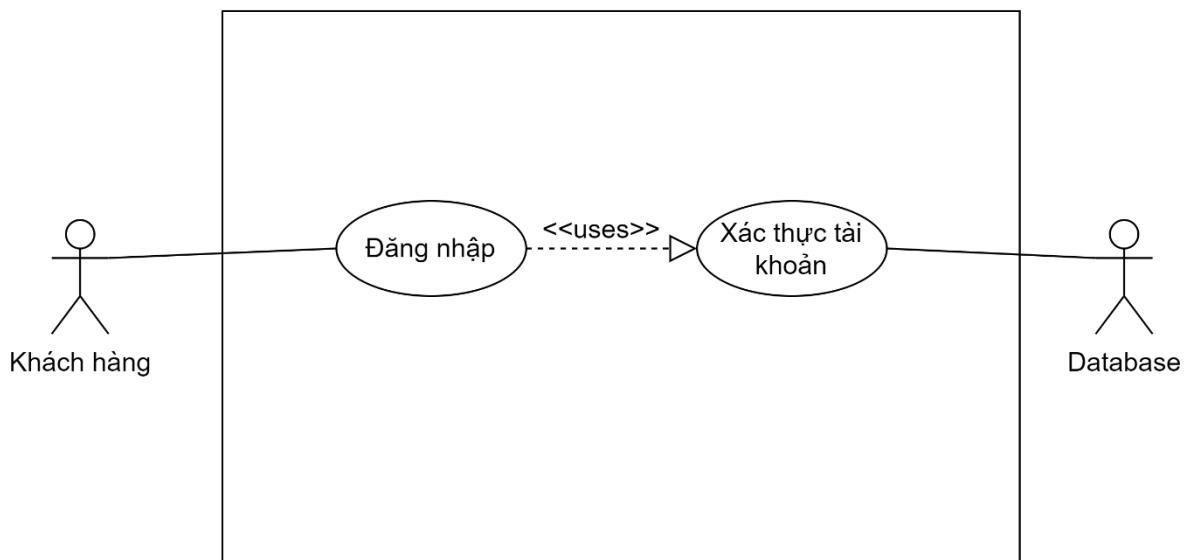
Đặc tả use case:

Tên Use case	Đăng ký
Tác nhân	Khách hàng, Database.
Mô tả	Cho phép khách hàng nhập thông tin để đăng ký tài khoản.
Điều kiện tiên quyết	Khách hàng tiến hành mở giao diện đăng ký.
Dòng sự kiện	<ul style="list-style-type: none">- Khách hàng nhập đầy đủ thông tin mà hệ thống yêu cầu nhập.- Hệ thống tiến hành kiểm tra thông tin.- Hệ thống lưu thông tin đăng ký, đưa ra thông báo đăng ký thành công và đưa khách hàng đến màn hình đăng nhập.- Kết thúc quá trình đăng ký.

Yêu cầu đặc biệt	<ul style="list-style-type: none"> - Thông tin về họ tên, email, mật khẩu, địa chỉ, giới tính cần phải chính xác. - Cần kết nối mạng internet.
Kết quả trả về	Thông báo khách hàng có đăng ký thành công hay không.

Bảng 4. Đặc tả use case đăng ký.

3.1.2.2. Use case đăng nhập.



Hình 18. Use case đăng nhập.

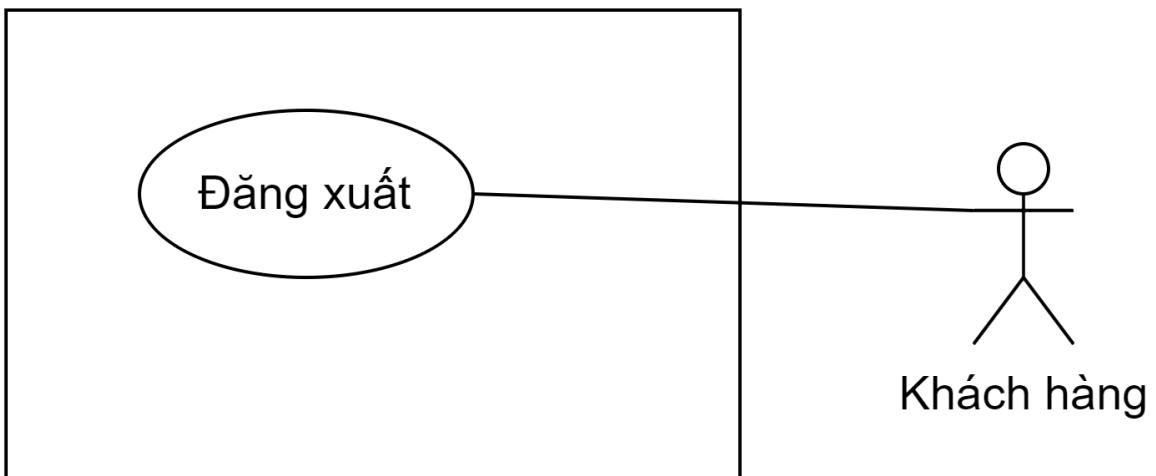
Đặc tả use case

Tên Use case	Đăng nhập hệ thống
Tác nhân	Khách hàng, Database.
Mô tả	Cho phép khách hàng đăng nhập vào hệ thống thực hiện chức năng đặt vé.
Điều kiện tiên quyết	Cần có tài khoản trong hệ thống và chưa đăng nhập vào hệ thống.
Dòng sự kiện	<ul style="list-style-type: none"> - Khách hàng tiến hành mở ứng dụng. - Khách hàng tiến hành điền mail và mật khẩu của mình vào khung đăng nhập. - Hệ thống tiến hành kiểm tra dữ liệu và xác minh thông tin khách hàng gửi vào. - Thông tin chính xác hệ thống sẽ đưa khách hàng truy cập vào hệ thống.

	<ul style="list-style-type: none"> - Nếu khách hàng đưa thông tin tài khoản hoặc mật khẩu không trùng với dữ liệu hệ thống. Hệ thống thông báo đăng nhập thất bại. - Kết thúc quá trình đăng nhập.
Yêu cầu đặc biệt	<ul style="list-style-type: none"> - Thông tin về email và mật khẩu yêu cầu phải chính xác. - Cần kết nối với mạng internet.
Kết quả trả về	Thông báo khách hàng có đăng nhập thành công hay không.

Bảng 5. Đặc tả use case đăng nhập.

3.1.2.3. Use case đăng xuất.



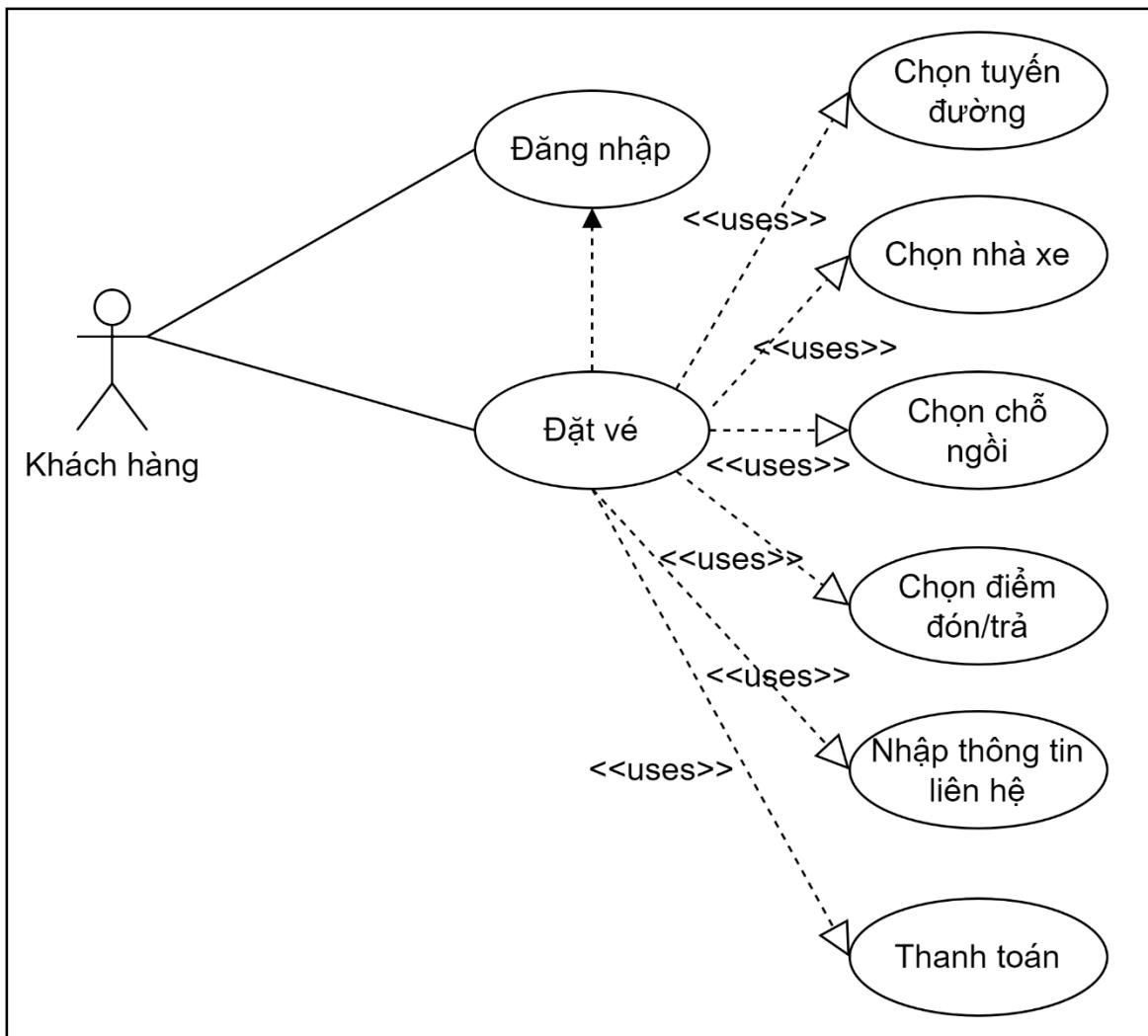
Hình 19. Use case đăng xuất.

Đặc tả use case

Tên Use case	Đăng xuất hệ thống
Tác nhân	Khách hàng.
Mô tả	Cho phép khách hàng đăng xuất khỏi hệ thống.
Điều kiện tiên quyết	Đã đăng nhập vào hệ thống.
Dòng sự kiện	<ul style="list-style-type: none"> - Khách hàng đã đăng nhập vào hệ thống. - Khách hàng xác nhận đăng xuất thì hệ thống sẽ trả về giao diện đăng nhập. - Kết thúc quá trình đăng nhập.
Yêu cầu đặc biệt	<ul style="list-style-type: none"> - Cần kết nối với mạng internet.
Kết quả trả về	Thoát khỏi hệ thống và thông báo đăng xuất thành công.

Bảng 6. Đặc tả use case đăng xuất.

3.1.2.4. Use case đặt vé.



Hình 20. Use case đặt vé.

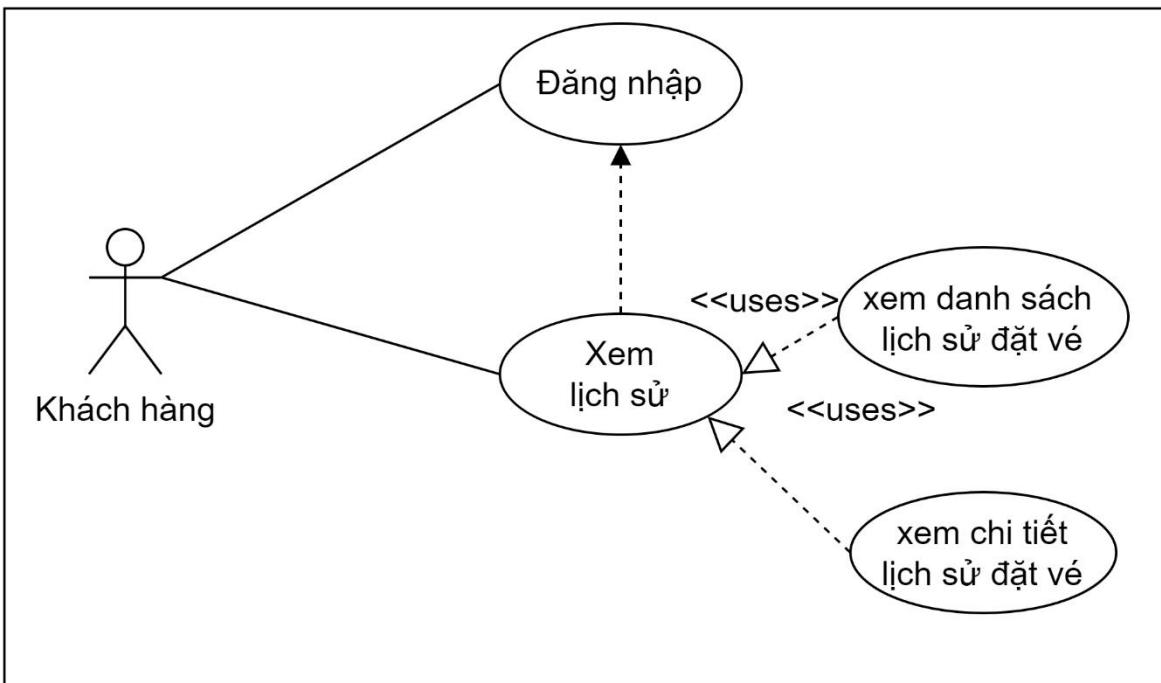
Đặc tả Use case

Tên Use case	Đặt vé
Tác nhân	Khách hàng.
Mô tả	Cho phép khách hàng thực hiện chọn tuyến đường, chọn nhà xe, chọn chỗ ngồi, chọn điểm đón/trả, nhập thông tin liên hệ.

Điều kiện tiên quyết	Đã đăng nhập tài khoản khách hàng
Dòng sự kiện	<ul style="list-style-type: none"> - Khách hàng tiến hành đăng nhập tài khoản của khách hàng. - Sau khi đăng nhập thành công sẽ hiển thị màn hình để chọn tuyến đường, chọn tỉnh thành đi và đến. - Tiếp đến là màn hình chọn nhà xe, hiển thị một danh sách để khách hàng chọn nhà xe phù hợp. - Tiếp đến là màn hình chọn ghế ngồi, khách hàng chọn ghế ngồi phù hợp. - Tiếp theo là màn hình chọn điểm đón/trả. - Tiếp đến là màn hình nhập thông tin liên hệ. - Cuối cùng là màn hình thông tin đặt vé, khách hàng kiểm tra lại thông tin.
Yêu cầu đặc biệt	<ul style="list-style-type: none"> - Cần kết nối với mạng internet.

Bảng 7. Đặc tả use case đặt vé.

3.1.2.5. Use case xem lịch sử



Hình 21. Use case xem lịch sử đặt vé.

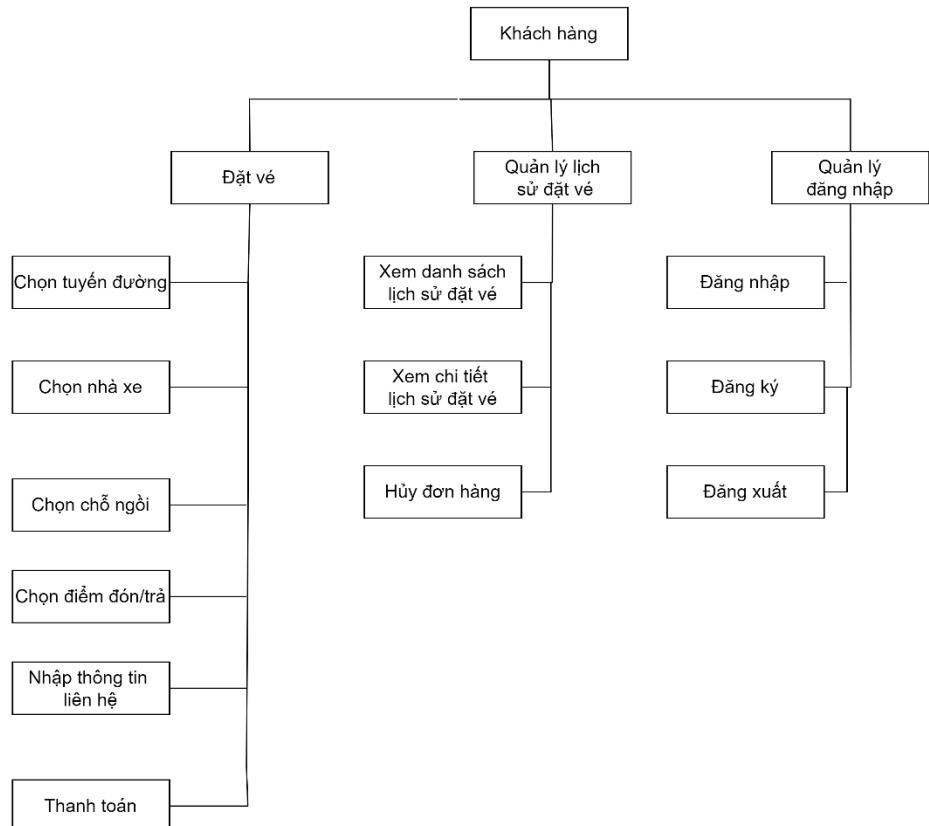
Đặc tả use case:

Tên Use case	Xem lịch sử đặt vé
Tác nhân	Khách hàng.
Mô tả	Cho phép khách hàng xem danh sách lịch sử và chi tiết đặt vé
Điều kiện tiên quyết	Đã đăng nhập tài khoản khách hàng
Dòng sự kiện	<ul style="list-style-type: none"> - Khách hàng tiến hành đăng nhập tài khoản - Khách hàng chọn chức năng lịch sử trên thanh bottom navigation - Hệ thống sẽ hiện ra danh sách lịch sử đặt vé

	<ul style="list-style-type: none"> - Khách hàng chọn lịch sử cần xem thông tin chi tiết và nhấn vào đó - Hệ thống sẽ hiển thị thông tin chi tiết lịch sử đặt vé
Yêu cầu đặc biệt	<ul style="list-style-type: none"> - Cần kết nối với mạng internet.

Bảng 8. Đặc tả use case xem lịch sử đặt vé.

3.2. Sơ đồ phân rã chức năng BFD

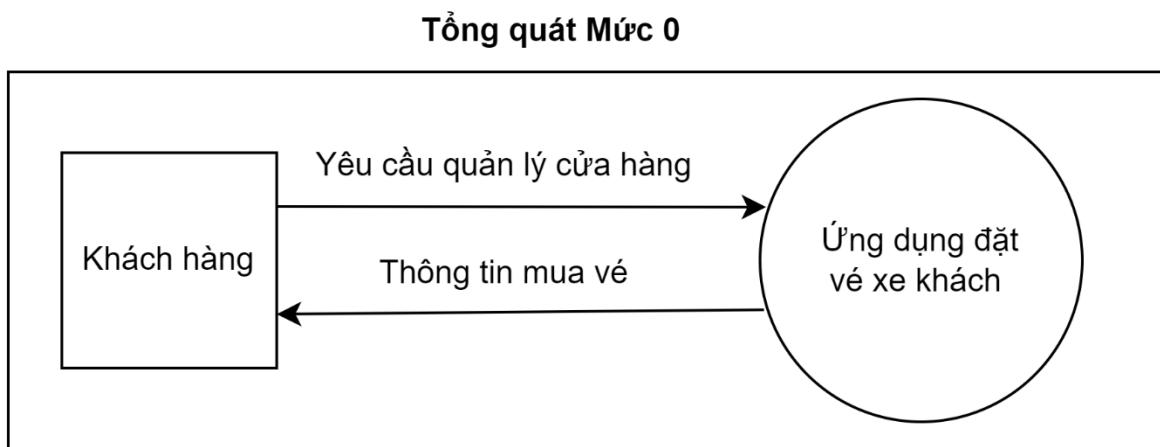


Hình 22. Sơ đồ phân rã chức năng BFD.

3.3. Sơ đồ DFD

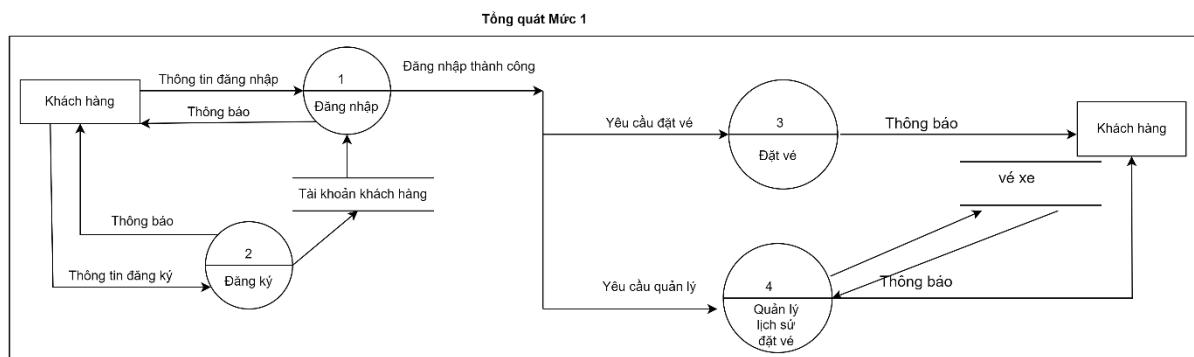
3.3.1. Sơ đồ DFD tổng quát:

Sơ đồ tổng quát mức 0



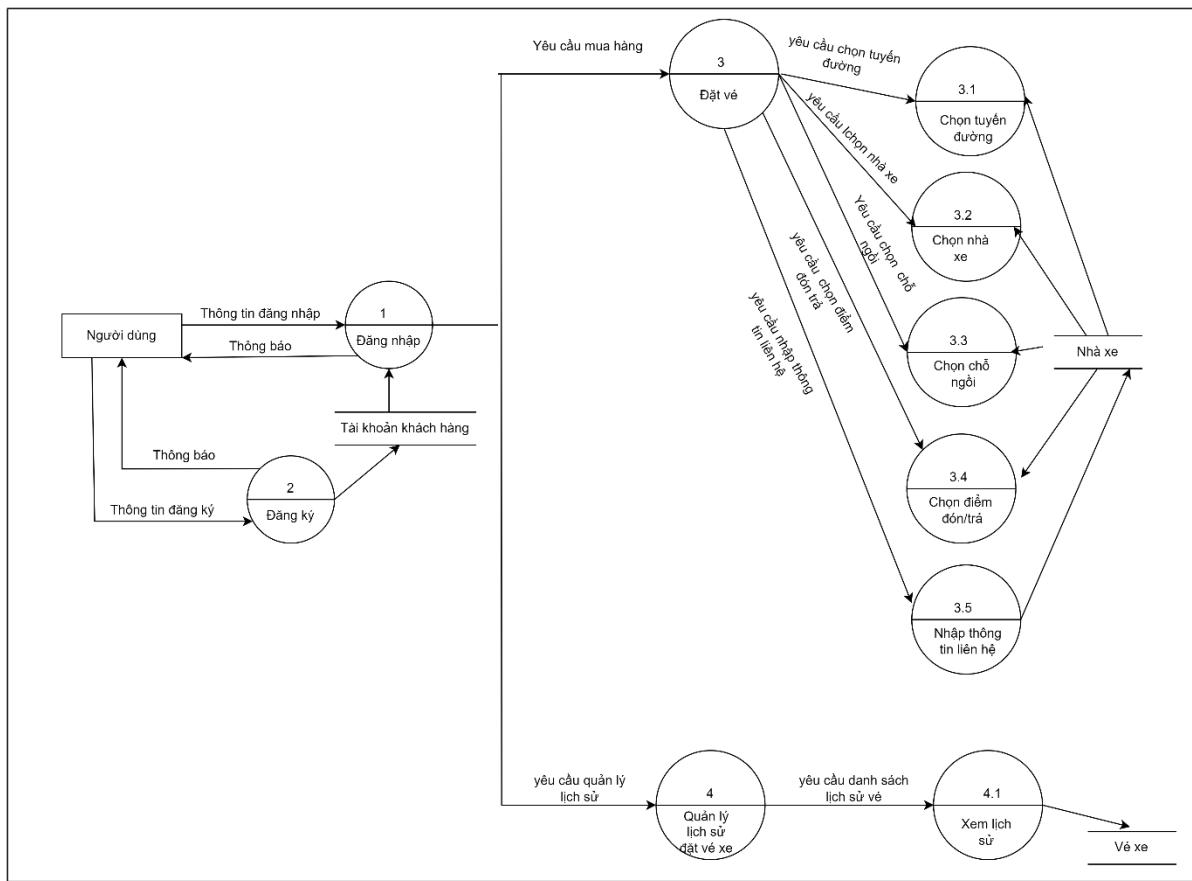
Hình 23. Sơ đồ DFD tổng quát mức 0.

Sơ đồ tổng quát mức 1



Hình 24. Sơ đồ DFD tổng quát mức 1.

Sơ đồ tổng quát mức 2

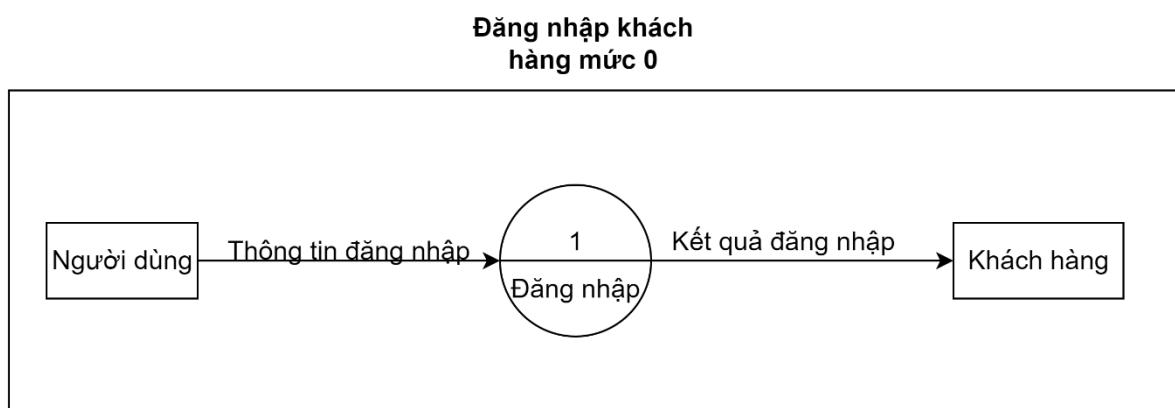


Hình 25. Sơ đồ DFD tổng quát mức 2.

3.3.2. Sơ đồ DFD chi tiết:

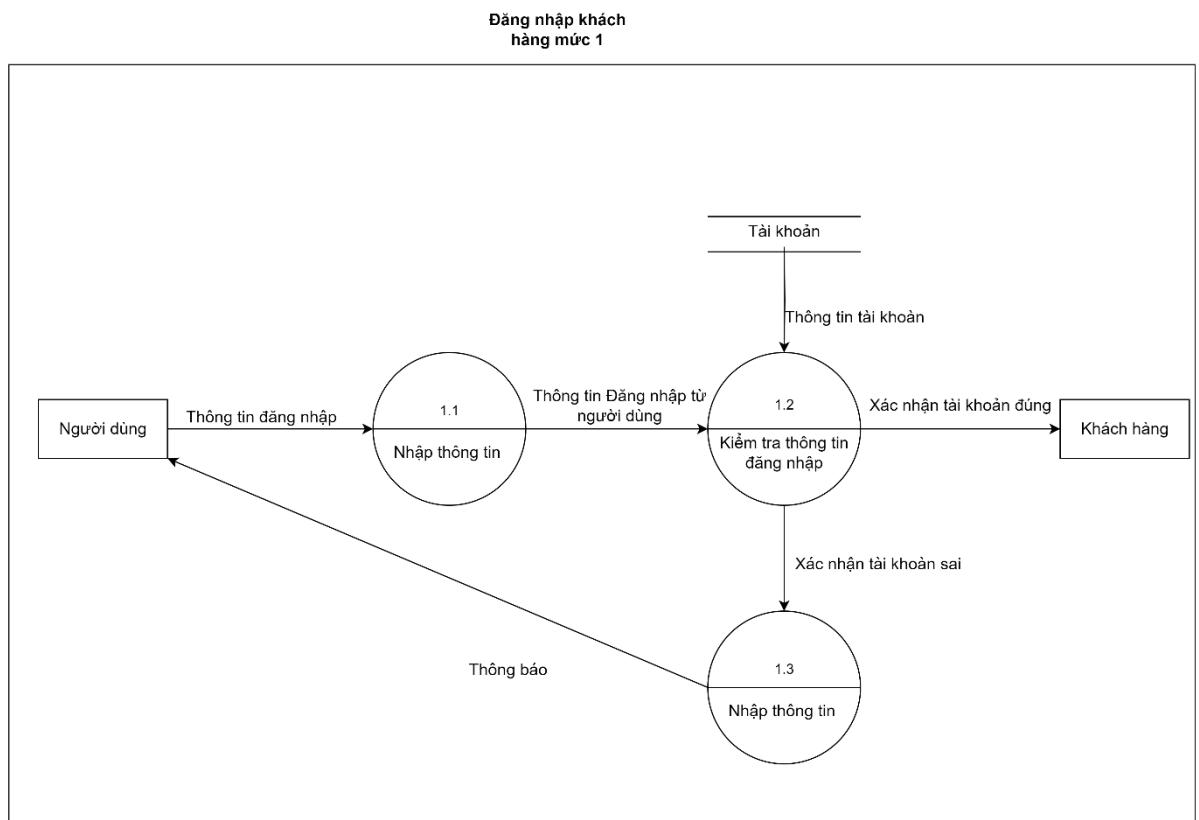
3.3.2.1. Sơ đồ DFD chức năng đăng nhập:

Sơ đồ DFD chức năng đăng nhập mức 0:



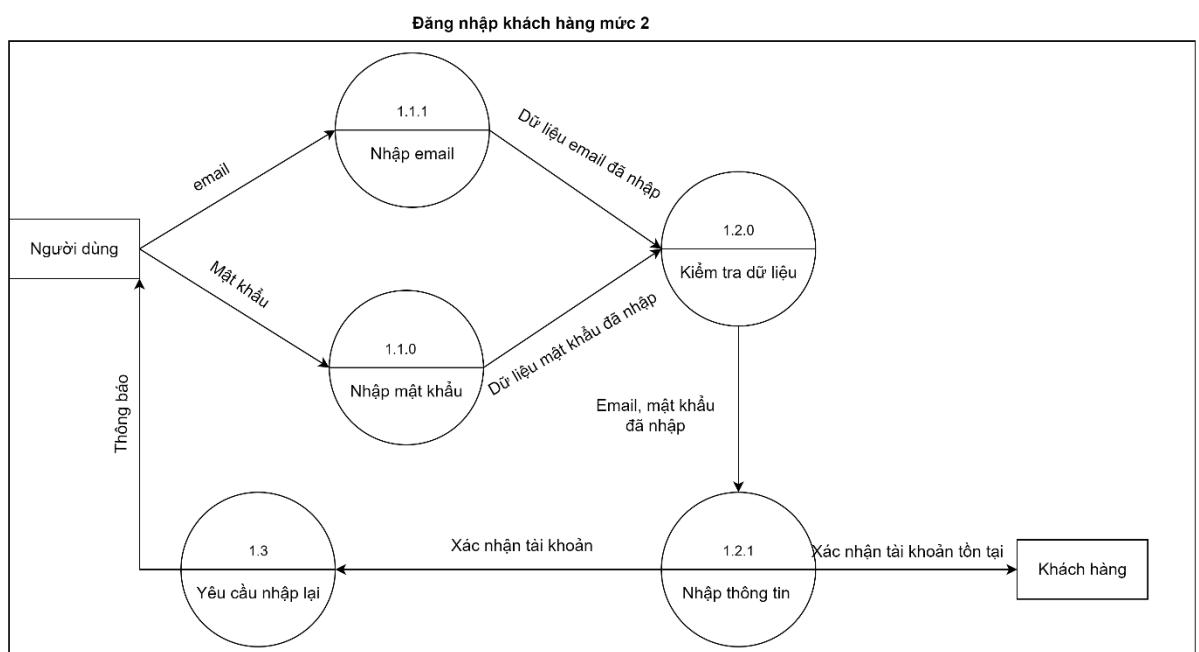
Hình 26. Sơ đồ DFD chức năng đăng nhập mức 0.

Sơ đồ DFD chức năng đăng nhập khách hàng mức 1:



Hình 27. Sơ đồ chức năng đăng nhập mức 1.

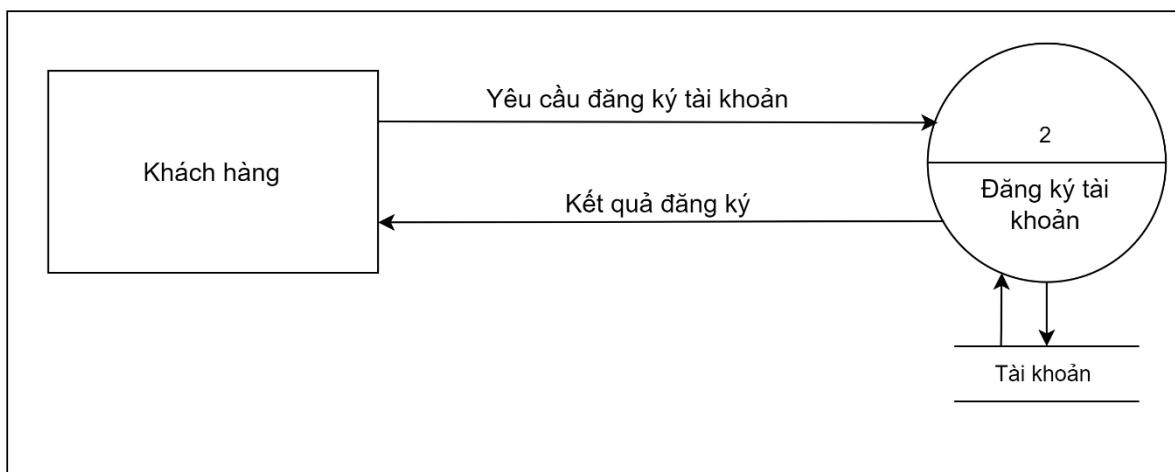
Sơ đồ DFD chức năng đăng nhập khách hàng mức 2:



Hình 28. Sơ đồ chức năng đăng nhập khách hàng mức 2.

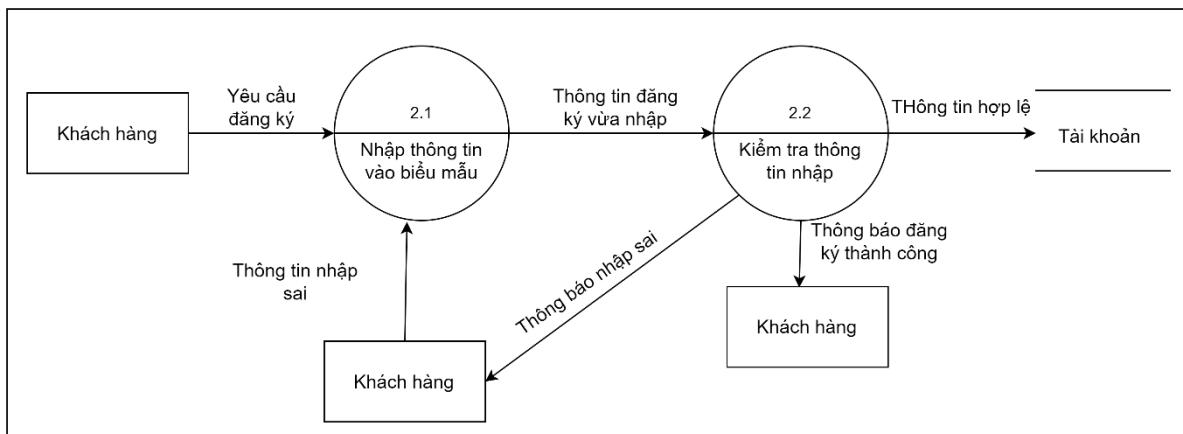
3.3.2.2. Sơ đồ DFD chức năng đăng ký:

Sơ đồ DFD chức năng đăng ký mức 0:



Hình 29. Sơ đồ DFD chức năng đăng ký mức 0.

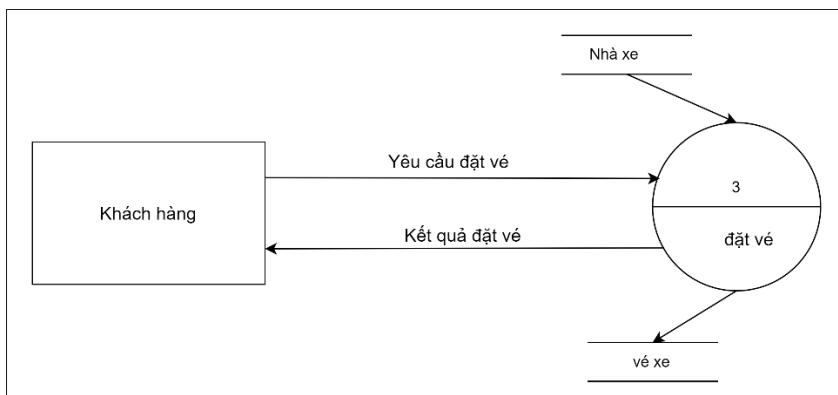
Sơ đồ DFD đăng ký chức năng mức 1:



Hình 30. Sơ đồ chức năng đăng ký mức 1.

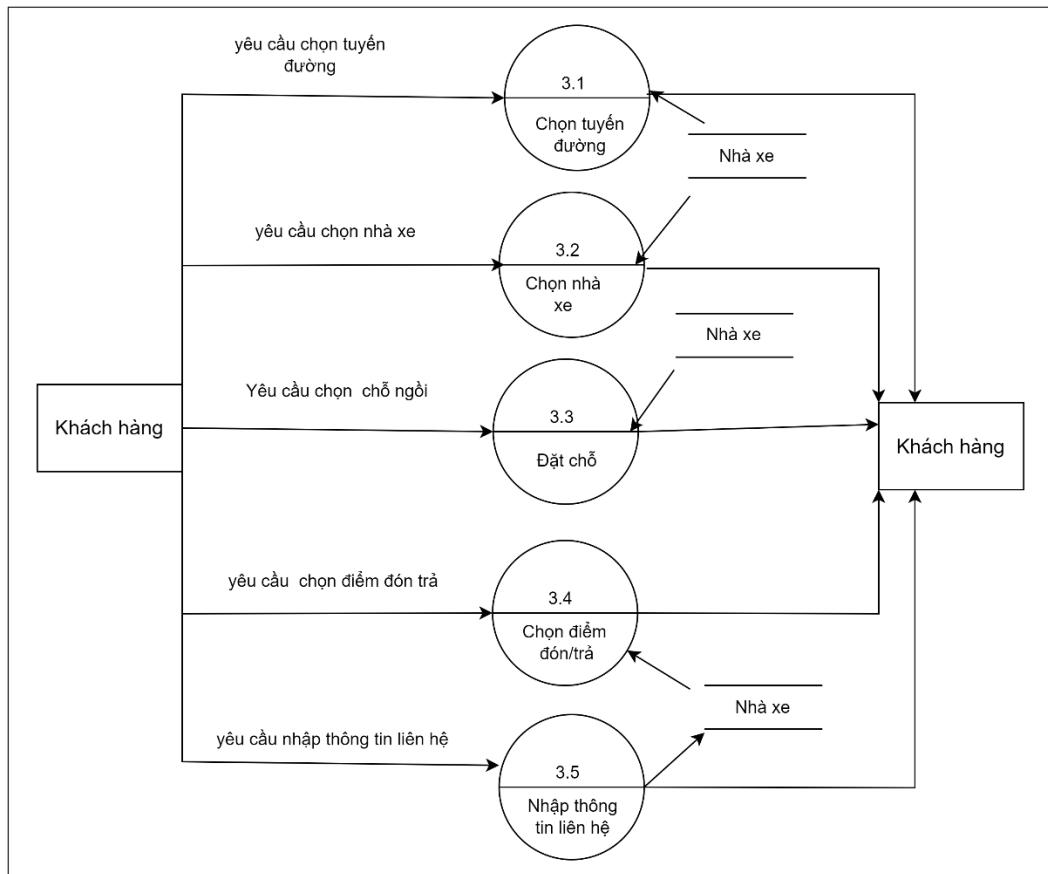
3.3.2.3. Sơ đồ DFD chức năng đặt vé xe:

Sơ đồ DFD chức năng đặt vé mức 0:



Hình 31. Sơ đồ DFD chức năng đặt vé mức 1.

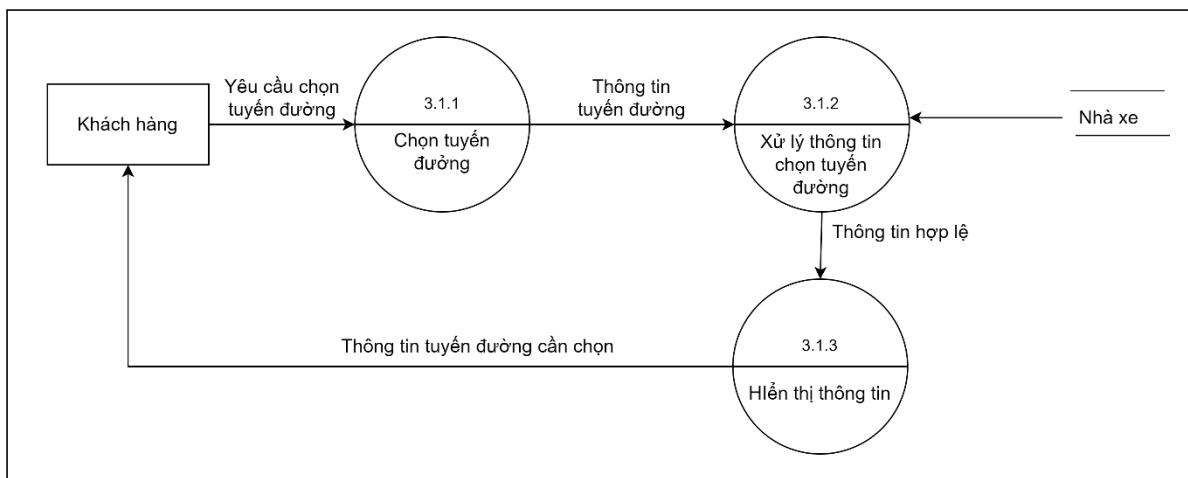
Sơ đồ DFD chức năng đặt vé mức 1:



Hình 32. Sơ đồ DFD chức năng đặt vé mức 1.

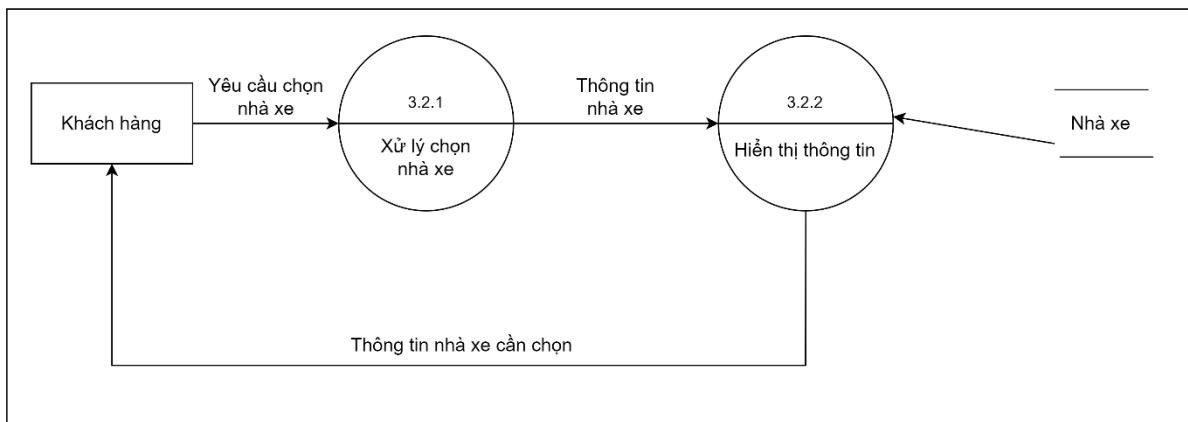
Sơ đồ DFD chức năng đặt vé mức 2:

Ô xử lý 3.1.



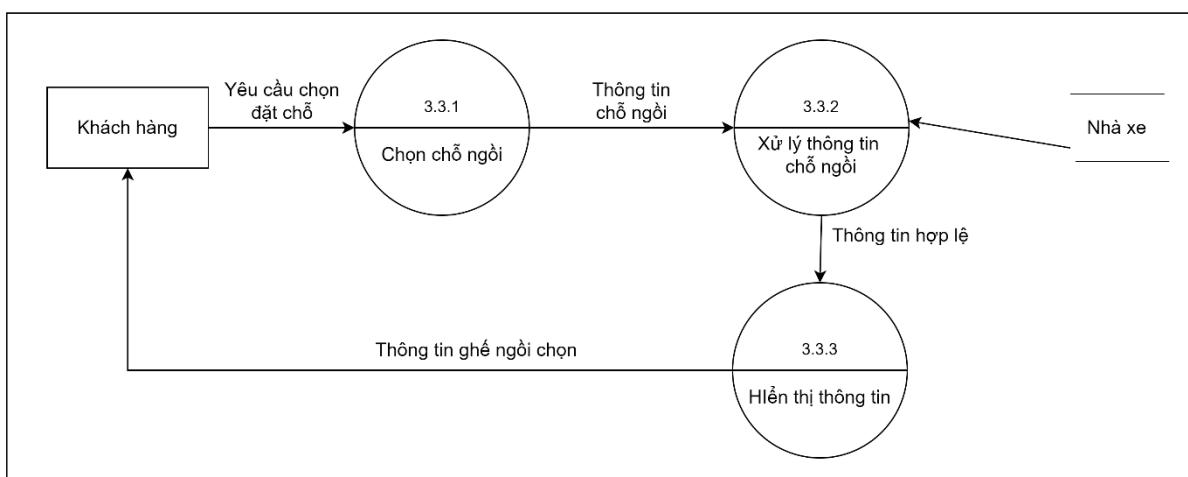
Hình 33. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.1.

Ô xử lý 3.2



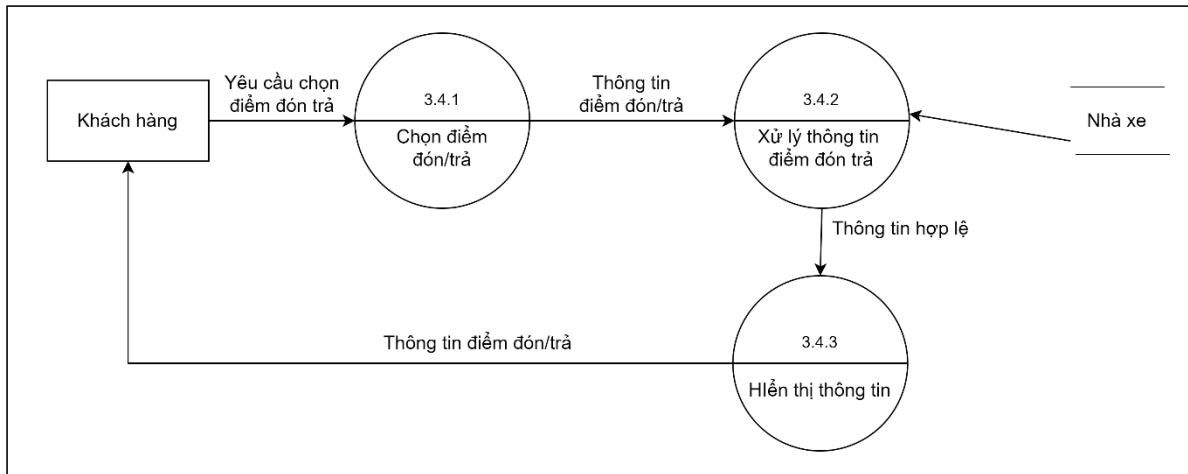
Hình 34. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.2.

Ô xử lý 3.3



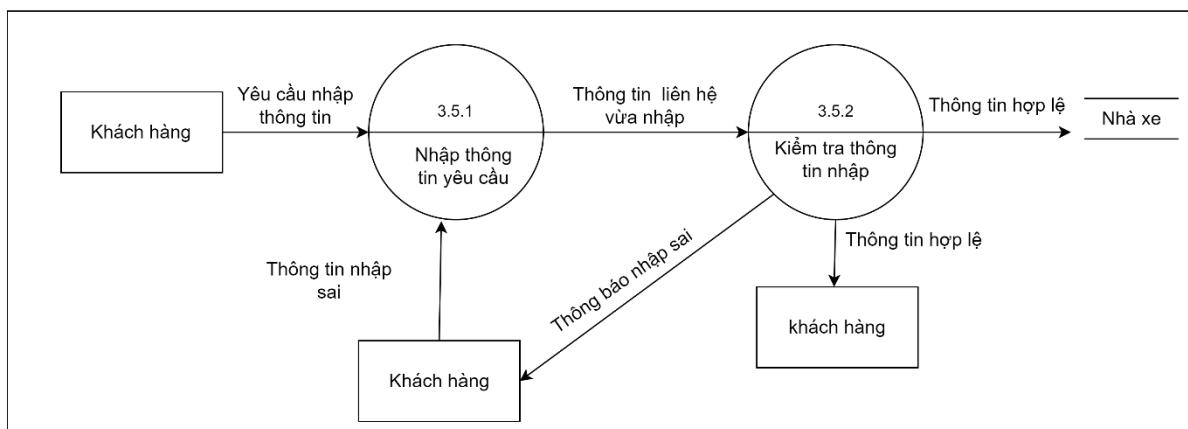
Hình 35. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.3.

Ô xử lý 3.4



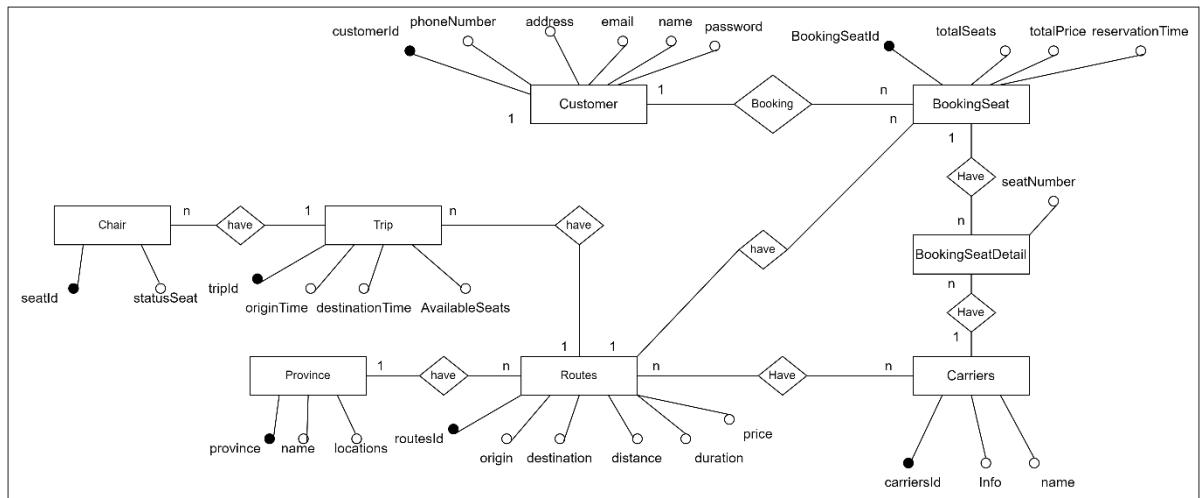
Hình 36. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.4.

Ô xử lý 3.5



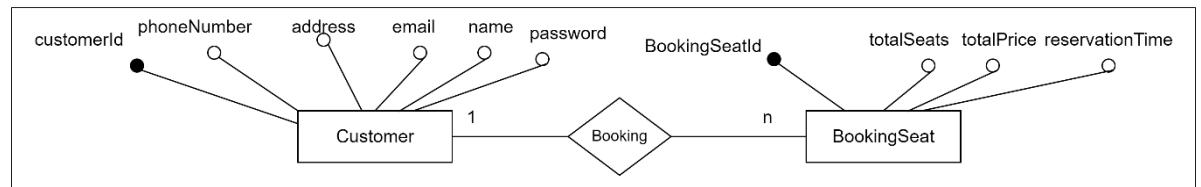
Hình 37. Sơ đồ DFD chức năng đặt vé mức 2 ô xử lý 3.5.

3.4. ERD.



Hình 38. ERD tổng quát.

3.4.1. ERD giữa Customer và BookingSeat.



Hình 39. ERD giữa Customer và BookingSeat.

Customer có 6 thuộc tính:

customerId: Mã khách hàng

phoneNumber: Số điện thoại

address: Địa chỉ

email: email khách hàng

name: Tên khách hàng

password: Mật khẩu

Trong đó, customerId: là mã định danh

BookingSeat có 5 thuộc tính:

bookingSeatId: Mã đặt chỗ

totalSeats: Tổng số lượng ghế đã đặt

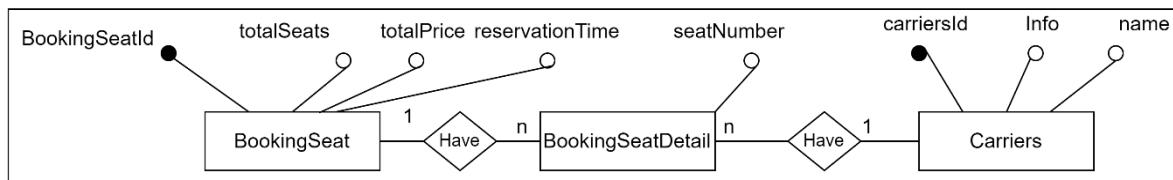
totalPrice: Tổng giá tiền

reservationTime: Thời gian đặt

Trong đó, bookingSeatId: là mã định danh

Customer và BookingSeat có mối quan hệ 1-n. Một khách hàng có thể đặt nhiều ghế ngồi, Một ghế ngồi chỉ thuộc một khách hàng.

3.4.2. ERD giữa BookingSeat, BookingSeatDetail và Carriers.



Hình 40. ERD giữa BookingSeat, BookingSeatDetail và Carriers.

BookingSeatDetail có một thuộc tính:

`seatNumber`: Số ghế cụ thể

Carriers có 3 thuộc tính

`carrierId`: Mã nhà xe

`info`: Thông tin nhà xe

`name`: Tên nhà xe

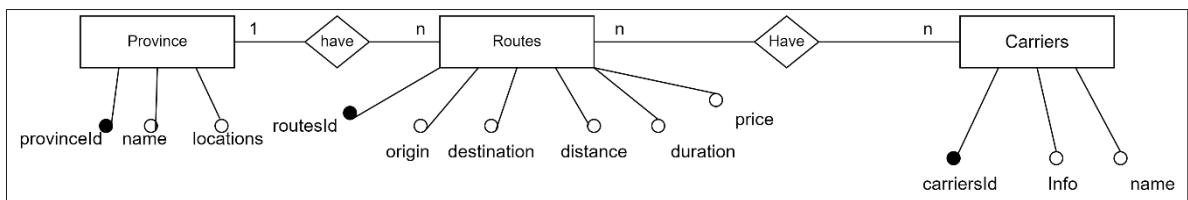
Trong đó, `carrierId`: là mã định danh

BookingSeatDetail là thực thể yếu sinh ra từ mối kết hợp n-n giữa BookingSeat và Carriers.

BookingSeat và BookingSeatDetail có mối kết hợp 1-n. Một đặt chỗ có thể có nhiều chi tiết đặt chỗ nhưng một chi tiết đặt chỗ chỉ thuộc về một đặt chỗ.

Carriers và BookingSeatDetail có mối kết hợp 1-n. Một nhà xe có nhiều chi tiết đặt chỗ, một chi tiết đặt chỗ chỉ thuộc về một nhà xe.

4.4.3. ERD giữa Province, Routes và Carriers.



Hình 41. ERD giữa Province, Routes và Carriers.

Province có 3 thuộc tính:

provinceId: Mã tỉnh thành

name: Tên tỉnh thành

locations: Các địa điểm trong một tỉnh thành

Trong đó, provinceId: là mã định danh.

Routes có 6 thuộc tính:

routesId: Mã tuyến đường

origin: Điểm đón của tuyến xe

destination: Điểm trả của tuyến xe

distance: Khoảng cách giữa các điểm

duration: Thời gian dự kiến của hành trình

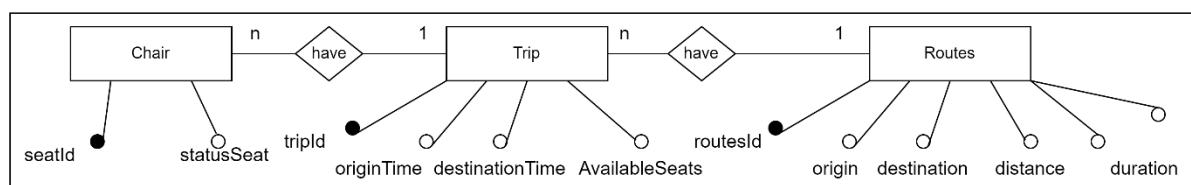
price: Giá vé

Trong đó, routesId là mã định danh

Province và Routes có mối kết 1-n. Một tỉnh thành có nhiều tuyến xe và một tuyến xe chỉ thuộc một tỉnh thành.

Routes và carrier có mối kết hợp n-n. Một tuyến xe có nhiều nhà xe và một nhà xe cũng có nhiều tuyến xe.

3.4.4. ERD giữa Chair, Trip và Routes



Hình 42. ERD giữa Chair, Trip và Routes.

Chair có 2 thuộc tính:

seatId: mã ghế ngồi

status: Trạng thái ghế ngồi

Trong đó, seatId là mã định danh

Trip có 4 thuộc tính

tripId: Mã Chuyến xe

originTime: Thời gian khởi hành

destinationTime: Thời gian đến

AvailableSeats: Số lượng ghế còn trống

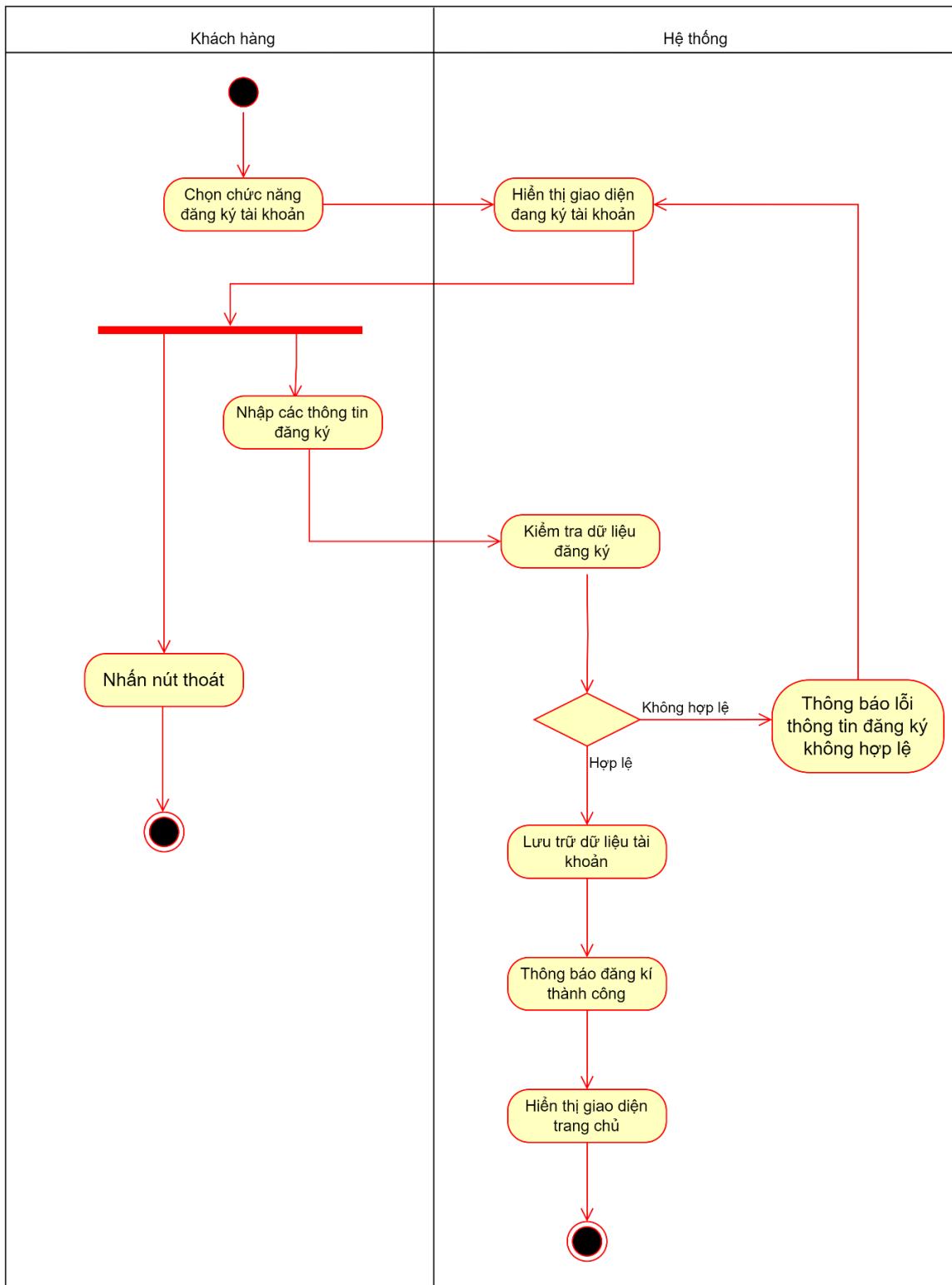
Trong đó, tripId là mã định danh

Trip và Chair có mối kết hợp 1-n. Một chuyến xe có nhiều ghế và một ghế chỉ thuộc về một chuyến xe.

Routes và Trip có mối quan hệ 1-n. Một tuyến xe có nhiều chuyến xe và một chuyến xe chỉ thuộc một tuyến xe.

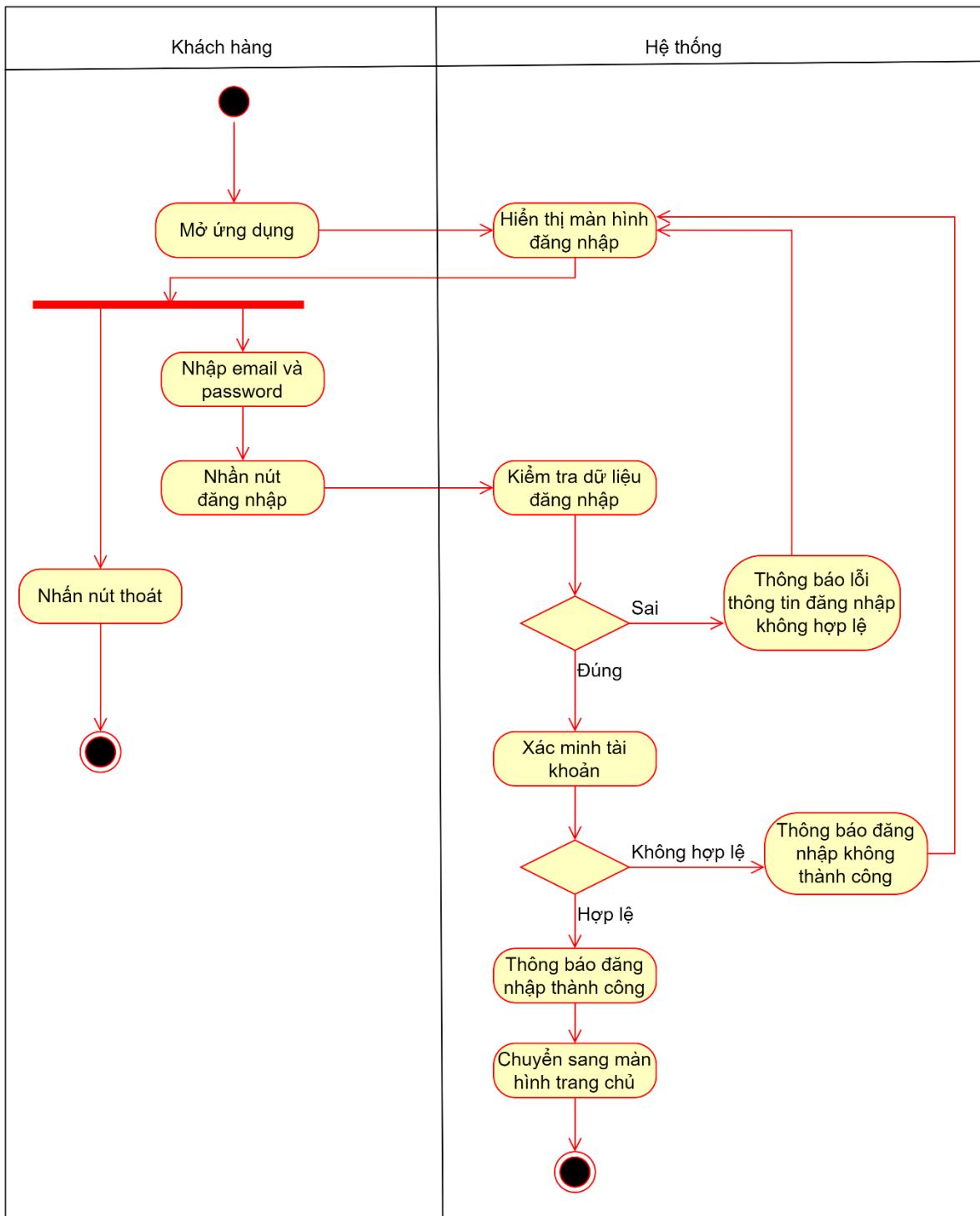
3.5. Activity

3.5.1. Activity đăng ký.



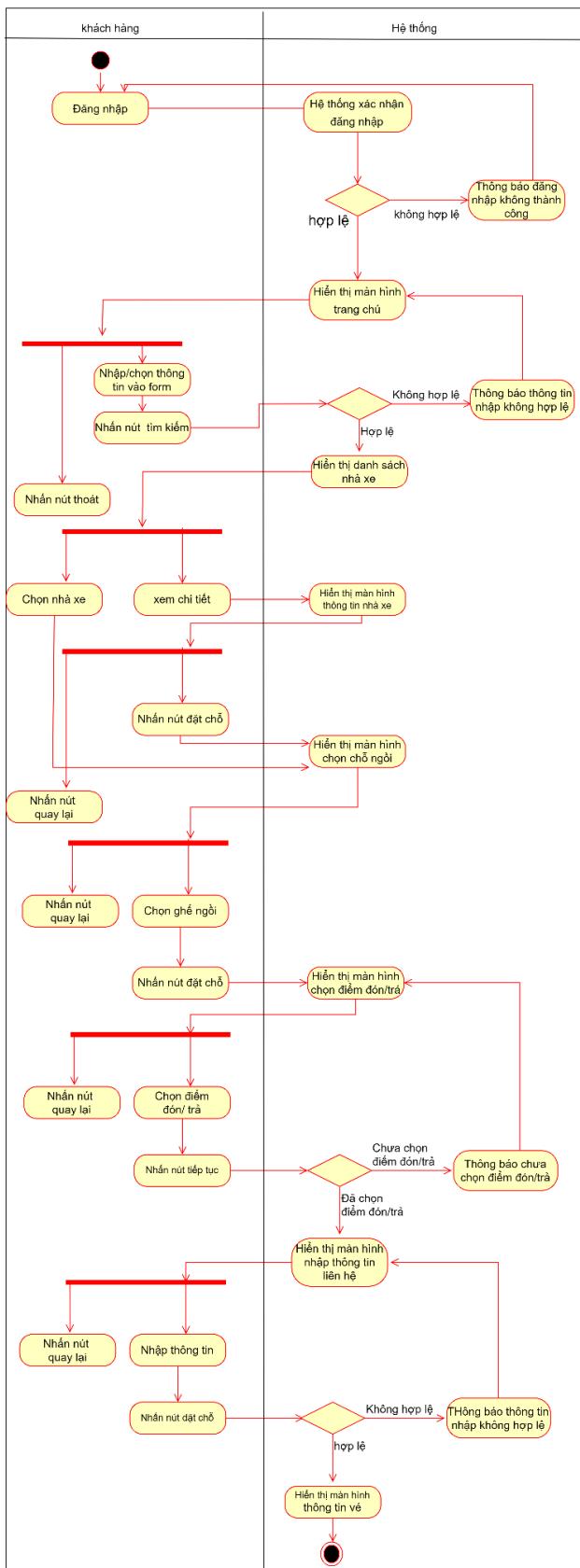
Hình 43. Activity đăng ký.

3.5.2. Activity đăng nhập.



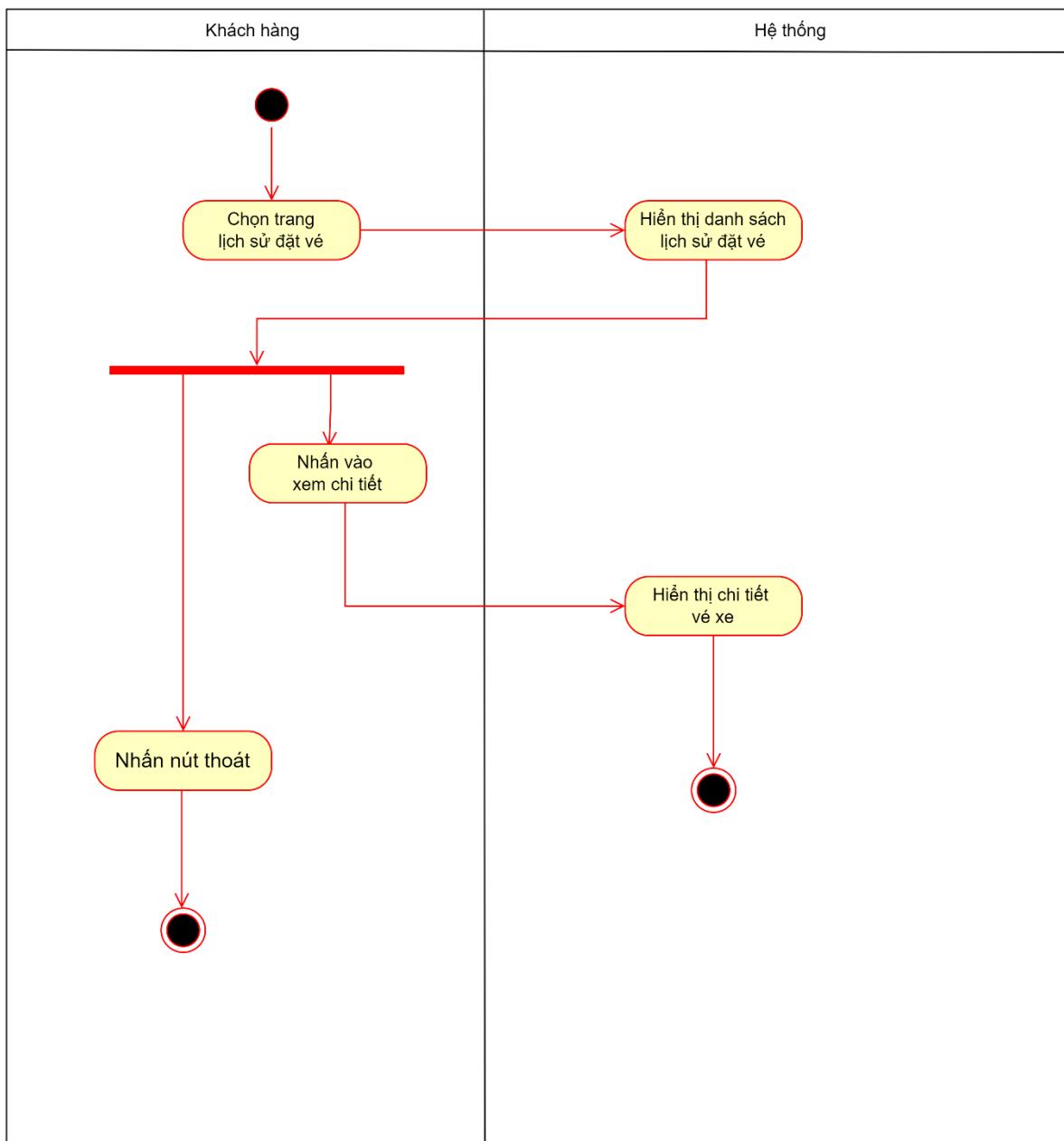
Hình 44. Activity đăng nhập.

3.5.3. Activity đặt vé



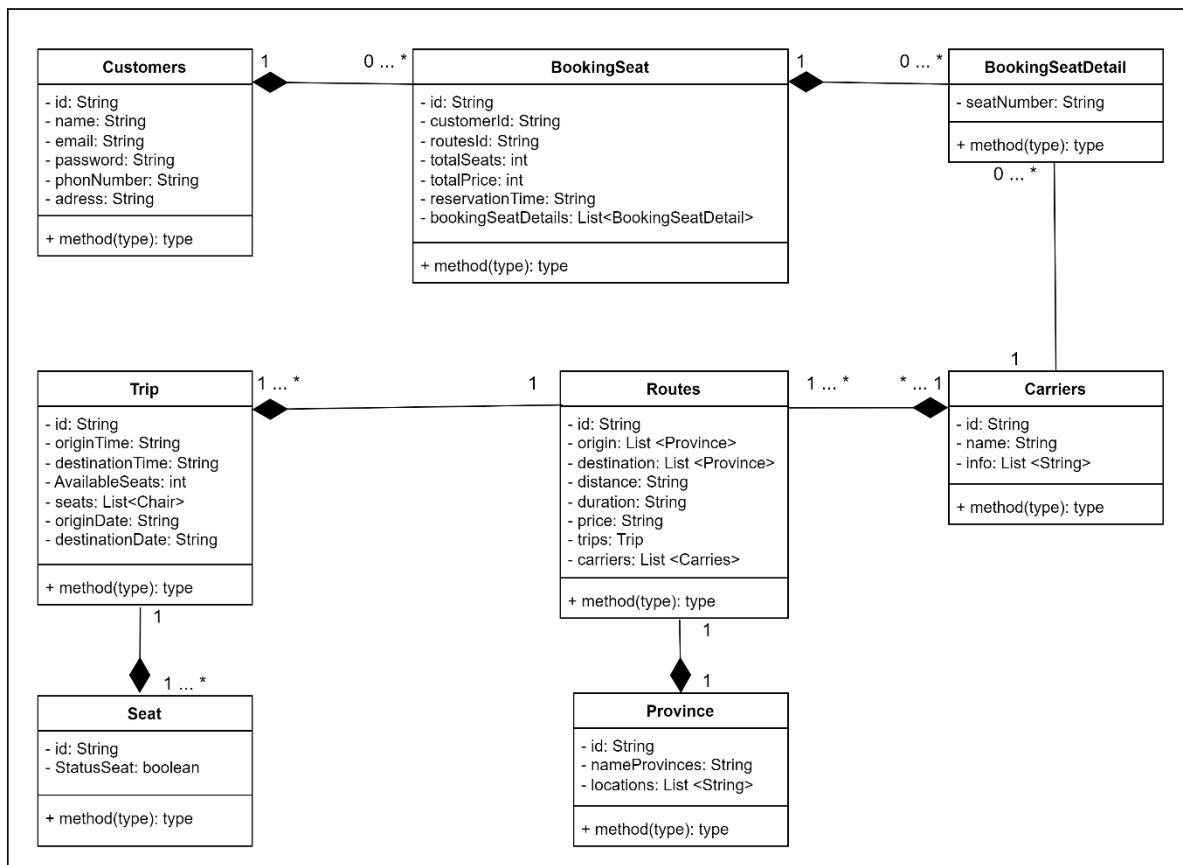
Hình 45. Activity đặt vé.

3.5.4. Activity xem lịch sử



Hình 46. Activity xem lịch sử đặt vé.

3.6. Biểu đồ lớp

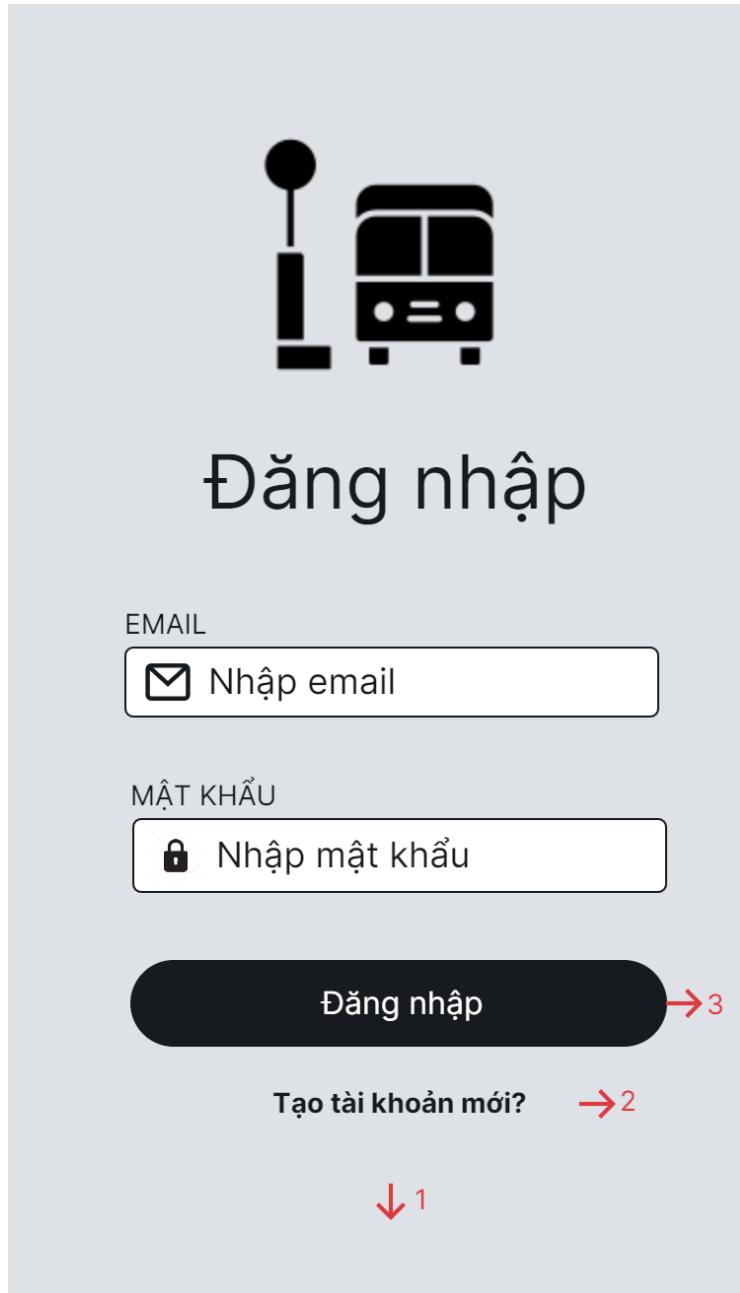


Hình 47. Biểu đồ lớp.

Chương 4: Thiết kế phần mềm

4.1. Thiết kế giao diện.

4.1.1. Thiết kế giao diện đăng nhập.



Hình 48. Thiết kế giao diện đăng nhập.

Bảng biến cố.

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khởi động ứng dụng	Hiển thị màn hình đăng nhập	
2	Khi ấn vào text chọn đăng ký tài khoản mới	Chuyển sang giao diện đăng ký tài khoản mới	
3	Khi ấn vào nút đăng nhập	Gọi hàm đăng nhập tài khoản với Email và mật khẩu	Email và mật khẩu phải hợp lệ

Bảng 9. Biến cố giao diện đăng nhập.

Mô tả giao diện đăng nhập khách hàng.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	imgIcon	ImageView	Hình ảnh biểu tượng của ứng dụng			
2	tvLogin	TextView	Tiêu đề trang đăng nhập			
3	tvEmail	TextView	Tên gọi của ô nhập email			
4	etEmail	EditText	Ô nhập Email đăng nhập			
5	tvPassword	TextView	Tên gọi của ô nhập mật khẩu			
6	etPassword	EditText	Ô nhập mật khẩu đăng nhập			

7	btnLogin	Button	Thực hiện quá trình đăng nhập			
8	btnSignUp	TextView	Nút đăng ký tài khoản mới			

Bảng 10. Mô tả giao diện đăng nhập.

4.1.2. Thiết kế giao diện đăng ký.



Đăng ký

1
↓

HỌ TÊN

EMAIL

MẬT KHẨU

ĐỊA CHỈ

SỐ ĐIỆN THOẠI

Đăng ký →²

Chưa có tài khoản? →³

Hình 49. Thiết kế giao diện đăng ký.

Bảng biến cố.

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Nhấn nút đăng ký từ giao diện đăng nhập	Hiển thị màn hình ký	
2	Khi nhấn vào nút đăng ký	Thực hiện quá trình đăng ký tài khoản	
3	Khi nhấn nút đã có tài khoản	Chuyển sang giao diện đăng nhập	

Bảng II. Biến cố giao diện đăng ký.

Mô tả giao diện đăng ký

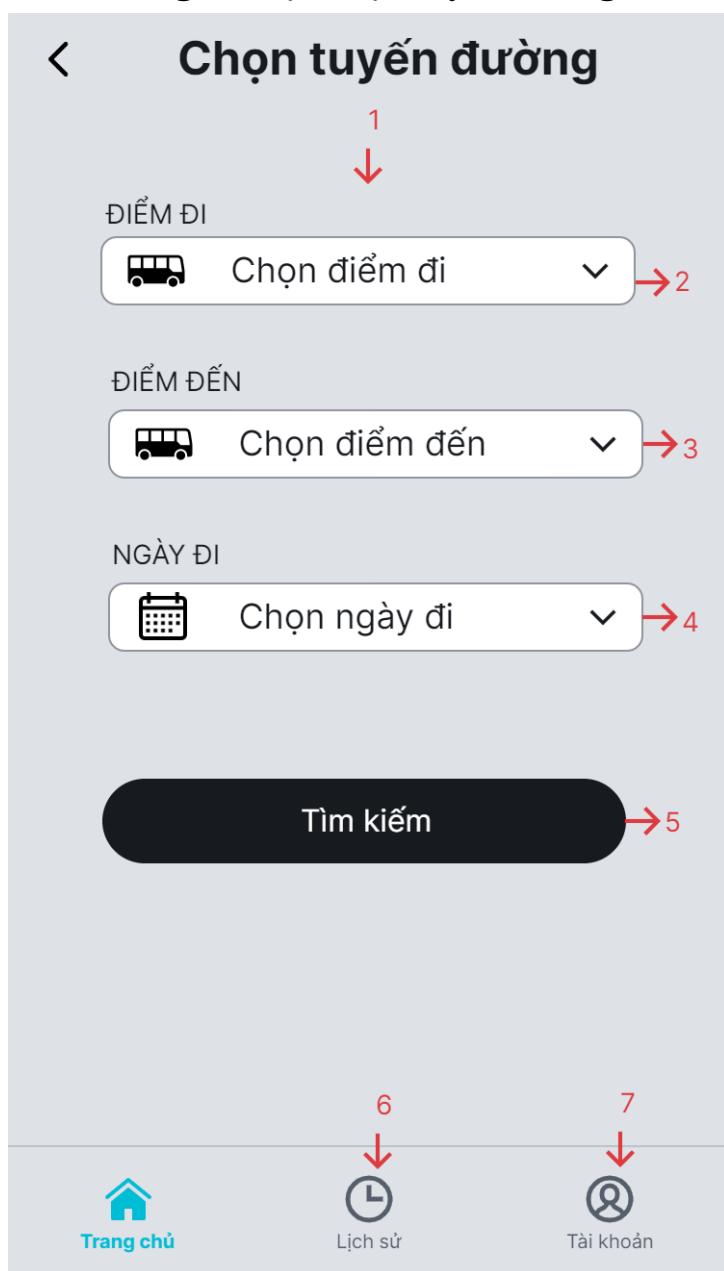
Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	imgIcon	ImageView	Hình ảnh biểu tượng của ứng dụng			
2	tvSignUp	TextView	Tiêu đề trang Signup			
3	etName	EditText	Ô nhập tên người dùng đăng ký			
4	etEmail	EditText	Ô nhập Email đăng ký			
5	etPassword	EditText	Ô nhập Password đăng ký			

6	btnMale	Button	Nút thay đổi giới tính male			
7	btnFemale	Button	Nút thay đổi giới tính female			
8	etAddress	EditText	Ô nhập địa chỉ đăng ký			
9	btnSignUp	TextView	Nút đăng ký tài khoản mới			
8	btnHaveAcc	TextView	Nút đã có tài khoản			

Bảng 12. Mô tả giao diện đăng ký.

4.1.3. Thiết kế giao diện đặt vé.

4.1.3.1. Thiết kế giao diện chọn tuyến đường



Hình 50. Thiết kế giao diện chọn tuyến đường.

Bảng biến cố

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khởi động màn hình	Hiển thị màn hình chọn tuyến đường	

2	Khi khách hàng nhấn nút lựa chọn điểm đi	Lấy danh sách điểm đi lựa chọn một trong những điểm đi đó	
3	Khi khách hàng nhấn nút lựa chọn điểm đến	Lấy danh sách điểm đến lựa chọn một trong những điểm đến đó	
4	Khi khách hàng chọn ngày đi	Hiển thị lịch cho khách hàng chọn	
5	Khi khách hàng nhấn nút thêm	Ghi data và chuyển sang màn hình tiếp theo	
6	Khi khách hàng nhấn bottom navigation lịch sử	Gọi hàm chuyển sang giao diện lịch sử.	
7	Khi khách hàng nhấn bottom navigation tài khoản	Gọi hàm chuyển sang giao diện tài khoản.	

Bảng 13. Biểu đồ giao diện chọn tuyến đường.

Mô tả giao diện chọn tuyến đường.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	tvSelectRoutes	TextView	Tiêu đề giao diện			

2	tvOrigin	TextView	Tên gọi của ô chọn điểm đi			
3	originSpiner	Spiner	Ô chọn điểm đi			
4	tvDestination	TextView	Tên gọi của ô chọn điểm đến			
5	destinationSpiner	Spiner	Ô chọn điểm đến			
6	tvReservationTime	TextView	Tên gọi của ô chọn ngày đi			
7	reservationTime	DateSpicker	Chọn ngày			
8	searchBtn	Button	Nút tìm kiếm tuyến đường			
9	history	Bottom navigation	Nút chuyển sang giao diện lịch sử			
10	account	Bottom navigation	Nút chuyển sang giao diện tài khoản			

Bảng 14. Mô tả giao diện chọn tuyến đường.

4.1.3.2. Thiết kế giao diện chọn nhà xe

< →¹ Chọn nhà xe →²

Hồ Chí Minh → Bình Định

Thứ 2, 25/09/2023

17:30	<u>12h46</u>	06:16	285.000đ
An Sương	660 km	An Nhơn	Còn 32 chỗ
Minh Hung			chi tiết → ³
17:30	<u>12h46</u>	06:16	285.000đ
An Sương	660 km	An Nhơn	Còn 28 chỗ
Minh Long			chi tiết
17:30	<u>12h46</u>	06:16	285.000đ
An Sương	660 km	An Nhơn	Còn 27 chỗ
Hoàng Hà			chi tiết
17:30	<u>12h46</u>	06:16	285.000đ
An Sương	660 km	An Nhơn	Còn 34 chỗ
Minh Long			chi tiết ↓ ⁴

Hình 51. Thiết kế giao diện chọn nhà xe.

Bảng biến cố

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khi nhấn nút quay về	Gọi hàm trả về giao diện trước đó	
2	Khi khách hàng nhấn nút lựa chọn chọn nhà xe	Lấy danh sách điểm đi lựa chọn một trong những nhà xe	

3	Khi khách hàng nhấn text chi tiết	Gọi hàm chuyển sang màn hình chi tiết thông tin nhà xe	
4	Khởi động màn hình	Hiển thị màn hình chọn nhà xe	

Bảng 15. Biểu đồ giao diện chọn nhà xe.

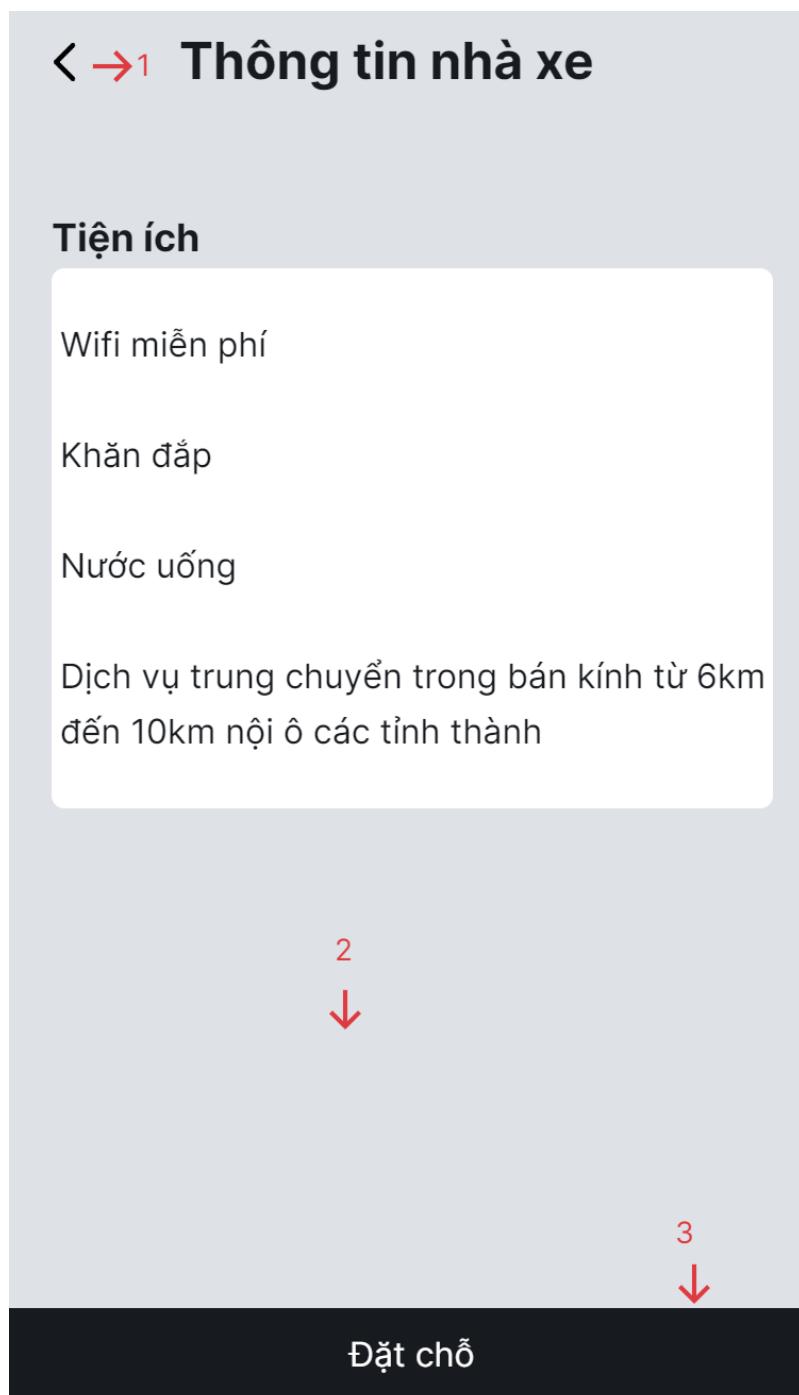
Mô tả giao diện chọn nhà xe.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	imageBack	ImageView	Nút quay về giao diện trước			
2	tvcarriers	TextView	Tiêu đề giao diện			
3	tvRoutes	TextView	Hiển thị tuyến đường			
4	tvreservationTime	TextView	Hiển thị thời gian đặt chỗ			
5	tvOriginTime	TextView	Hiển thị thời gian xuất phát			

6	tvLocationOrigin	TextView	Hiển thị địa điểm đón			
7	tvDuration	TextView	Hiển thị thời gian			
8	tvDistance	TextView	Hiển thị khoảng cách			
9	tvDestinationTime	TextView	Hiển thị thời gian đến			
10	tvLocationDestination	TextView	Hiển thị địa điểm đến			
11	tvCarriers	TextView	Hiển thị tên nhà xe			
12	tvDetail	TextView	textView chuyển đến màn hình khác			

Bảng 16. Mô tả giao diện chọn nhà xe.

4.1.3.3. Thiết kế giao diện thông tin nhà xe



Hình 52. Thiết kế giao diện chọn nhà xe.

Bảng biến cố

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khi nhấn nút quay về	Gọi hàm trả về giao diện trước đó	
2	Khởi động màn hình	Hiển thị màn hình chọn nhà xe và đọc dữ liệu tiện ích nhà xe	
3	Khi nhấn vào nút đặt chỗ	Gọi hàm chuyển sang giao diện chọn ghế ngồi	

Bảng 17. Biến cố giao diện thông tin nhà xe.

Mô tả giao diện thông tin nhà xe.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	imageBack	ImageView	Nút quay về giao diện quản trị			
2	tvInfo	TextView	Tiêu đề giao diện			
3	tvTitleUtilities	TextView	Tên gọi của ô thông tin tiện ích			
4	tvUtilites	TextView	Hiển thị thông tin tiện ích			

5	BookingSeatBtn	Button	Nút đặt chỗ chuyển sang giao diện khác			
---	----------------	--------	---	--	--	--

Bảng 18. Mô tả giao diện thông tin nhà xe.

4.1.3.4. Thiết kế giao diện chọn ghế.

Chọn chỗ ngồi

< → 1 2 ↓

Ghế đã chọn Ghế đang chọn 285.000đ/ghế

Tang 1			Tang 2		
A01	A02	A03	B01	B02	B03
A04	A05	A06	B04	B05	B06
A07	A08	A09	B07	B08	B09
A10	A11	A12	B10	B11	B12
A13	A14	A15	B13	B14	B15
A16		A17	B16		B17
A18	A19	A20	A21	A22	B18 B19 B20 B21 B22

3 ↑

Mã ghế 4 Tổng tiền

A12 ↓ 285.000đ

Tiếp tục

Hình 53. Thiết kế giao diện chọn chỗ ngồi.

Bảng biến cố

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khi nhấn nút quay về	Gọi hàm trả về giao diện trước đó	
2	Khởi động màn hình	Hiển thị màn hình chọn nhà xe và đọc dữ liệu trạng thái ghế nhà xe	
3	Khi nhấn chọn ghế	Hiển thị trạng thái được chọn và ghi dữ liệu	
4	Khi nhấn vào nút đặt chỗ	Gọi hàm chuyển sang giao diện mới.	

Bảng 19. Biến cố giao diện chọn ghế.

Mô tả giao diện chọn ghế.

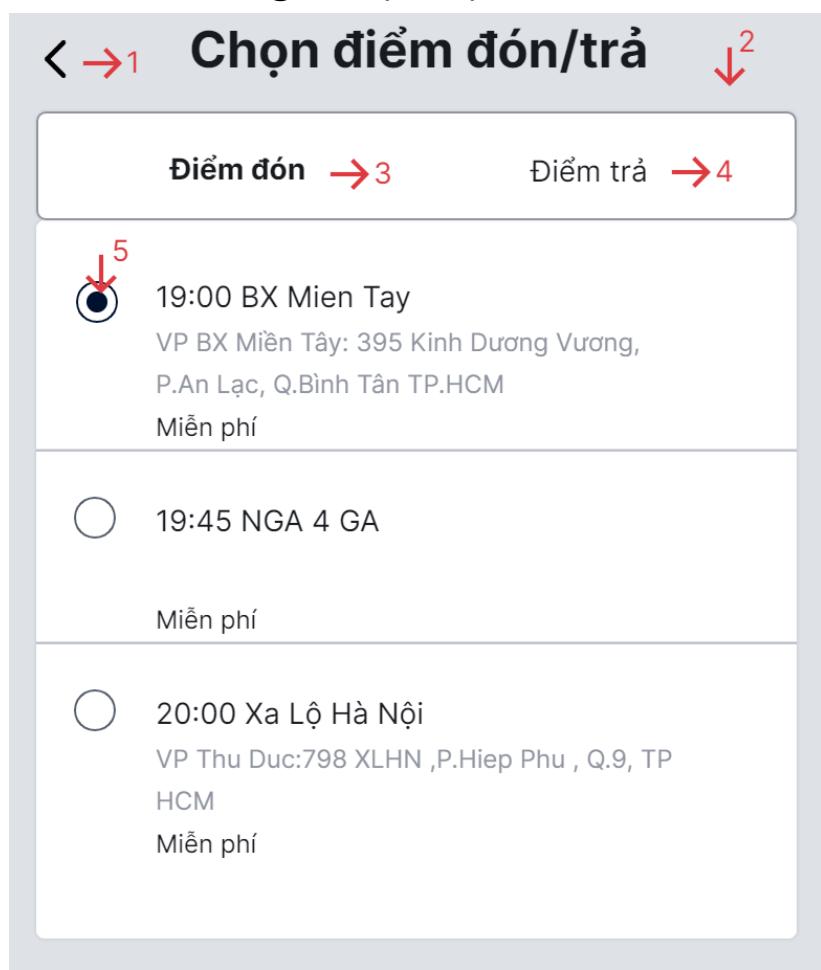
Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	imageBack	ImageView	Nút quay về giao diện quản trị			
2	tvTitle	TextView	Tiêu đề giao diện			

3	imageSlected	ImageView	Icon thể hiện trạng thái ghế đã chọn			
4	tvSelected	TextView	Mô tả icon trạng thái đã chọn			
5	imageSlecting	ImageView	Icon thể hiện trạng thái ghế đang chọn			
6	tvSelecting	TextView	Mô tả icon trạng thái đang chọn			
7	imageNotSlect	ImageView	Icon thể hiện trạng thái ghế chưa chọn			
8	tvNotSelect	TextView	Mô tả icon trạng thái chưa chọn và giá tiền			
9	seatBtn	Button	Nút chọn ghế			
10	tvNameSeatCode	TextView	Tên gọi của ô hiển thị mã ghế			
11	tvSeatCode	TextView	Mã ghế			

12	tvNameTotalPrice	TextView	Tên gọi của ô hiển thị tổng tiền			
13	tvTotalPrice	TextView	Hiển thị tổng tiền			
14	NextBtn	Button	Nút chuyển sang giao diện mới			

Bảng 20. Mô tả giao diện chọn ghế.

4.1.3.5. Thiết kế giao diện chọn điểm đón/điểm trả



Điểm đón

BX Mien Tay

Miễn phí

Điểm trả

BX An Nhon

Miễn phí

6



Tiếp tục

Hình 54. Thiết kế giao diện chọn điểm đón, điểm trả.

Bảng biến cố

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
-----	---------------------	-------	---------

1	Khi nhấn nút quay về	Gọi hàm trả về giao diện trước đó	
2	Khởi động màn hình	Hiển thị màn hình chọn nhà xe và đọc dữ liệu từ server	
3	Khi textView điểm đón	Đọc dữ liệu từ database và hiện thị danh sách điểm đón	
4	Khi textView điểm trả	Đọc dữ liệu từ database và hiện thị danh sách điểm trả	
5	Khi nhấn vào radioButton	Thực hiện ghi dữ liệu	
6	Khi nhấn Button tiếp tục	Gọi hàm chuyển sang màn hình mới	

Bảng 21. Biến cờ giao diện chọn điểm đón, điểm trả.

Mô tả giao diện chọn điểm đón/trả.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	imageBack	ImageView	Nút quay về giao diện quản trị			
2	tvTitile	TextView	Tiêu đề giao diện			

3	tvLocationOrigin	TextView	Nút chuyển sang mà hình chọn điểm đón			
4	tvLocationDestination	TextView	Nút chuyển sang mà hình chọn điểm trả			
5	radioBtnSelectLocation	RadioButton	Nút chọn vị trí điểm đón/trả			
6	textInfoOrigin	TextView	Hiển thị thông tin điểm đón/trả			
7	nextBtn	Button	Chuyển sang màn hình mới			

Bảng 22. Mô tả giao diện chọn điểm đón, điểm trả.

4.1.3.6. Thiết kế giao diện thông tin liên hệ

← →¹ **Thông tin liên hệ**

Họ và tên
Nguyễn Văn An

email
nguyenvanan@gmail.com

Số điện thoại
0347398738

2
↓

3
↓

Xác nhận đặt Chỗ

Hình 55. Thiết kế giao diện thông tin liên hệ.

Bảng biến cố

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khi nhấn nút quay về	Gọi hàm trả về giao diện trước đó	
2	Khởi động màn hình	Hiển thị màn hình chọn nhà xe và đọc dữ liệu từ server	
3	Khi nhấn nút xác nhận đặt chỗ	Ghi dữ liệu và gọi hàm chuyển sang màn hình mới	

Bảng 23. Biểu đồ giao diện thông tin liên hệ.

Mô tả giao diện thông tin liên hệ.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	imageBack	ImageView	Nút quay về giao diện quản trị			
2	tvTitle	TextView	Tiêu đề giao diện			
3	tvName	TextView	Tên gọi của ô nhập họ tên			
4	nameEt	EditText	Ô nhập họ tên			
5	tvEmail	TextView	Tên gọi của ô nhập họ tên			
6	emailEt	EditText	Ô nhập email			
7	tvPhoneNuber	TextView	Tên gọi của ô nhập số điện thoại			
8	phoneNumberEt	EditText	Ô nhập số điện thoại			
9	confirmBtn	Button	Chuyển sang màn hình mới			

Bảng 24. Mô tả giao diện thông tin liên hệ.

4.1.3.7. Thiết kế giao diện thông tin đặt vé.

Thông tin đặt vé

Thông tin chuyến đi

Minh Hùng
Mã ghế: A12

17:30 25/09/2023 12h46 (Du kien)	Ho Chi Minh VP BX Miền Tây: 395 Kinh Dương Vương, P.An Lạc, Q.Bình Tân TP.HCM
06:16 26/09/2023	Bình Dinh An Nhơn - 02 Nguyễn Văn Linh, P.Bình Định, Tx.An

Thông tin vé

Nguyễn Văn An	Số điện thoại: 0347390730
Email: nguyenvanan@gmail.com	
Vé	
1 x Vé 285.000đ	285.000đ

Tổng tiền: **285.000đ**

Thanh toán

Hình 56. Thiết kế giao diện thông tin đặt vé.

Bảng biến cỗ

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khi nhấn nút quay về	Gọi hàm trả về giao diện trước đó	

2	Khởi động màn hình	Hiển thị màn hình chọn nhà xe và đọc dữ liệu từ database	
3	Khi nhấn nút thanh toán	Thực hiện thanh toán	

Bảng 25. Biểu đồ giao diện thông tin đặt vé.

Mô tả giao diện thông tin đặt vé.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	imageBack	ImageView	Nút quay về giao diện quản trị			
2	tvTitile	TextView	Tiêu đề giao diện			
3	tvInfoTrip	TextView	Tiêu đề thông tin chuyến đi			
4	tvCarriers	TextView	Tên nhà xe			
5	tvSeatCode	TextView	Mã ghế			
6	textInfoOrigin	TextView	Hiển thị điểm đón/trả			
7	tvDuration	TextView	Hiển thị thời gian			

8	tvOriginTime	TextView	Hiển thị thời gian xuất phát			
9	tvDestinationTime	TextView	Hiển thị thời gian đến			
10	tvInfoTicket	TextView	Tiêu đề thông tin vé			
11	tvName	TextView	Hiển thị tên khách hàng			
12	tvPhoneNumber	TextView	Hiển thị số điện thoại			
13	tvEmail	TextView	Hiển thị mail			
14	tvTotalPrice	TextView	Hiển thị tổng tiền			
15	payBtn	Button	Nút thanh toán			

Bảng 26. Mô tả giao diện thông tin đặt vé.

4.1.4. Thiết kế giao diện lịch sử đặt vé.

Lịch sử đặt vé

F5686999jghn
Hồ Chí Minh → Bình Định
25/09/2023
285.000đ 1 → chi tiết

F5686999jghn
Hồ Chí Minh → Bình Định
25/09/2023
285.000đ chi tiết

F5686999jghn
Hồ Chí Minh → Bình Định
25/09/2023
285.000đ chi tiết

2
↓
3
↓
Trang chủ Lịch sử Tài khoản
4
↓

Hình 57. Thiết kế giao diện xem lịch sử đặt vé.

Bảng biến cố

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khi nhấn vào chi tiết	Gọi hàm chuyển sang màn hình chi tiết	

2	Khởi động màn hình	Hiển thị màn hình lịch sử và đọc dữ liệu từ database	
3	Khi nhấn bottom trang chủ	Hiển thị màn hình về trang chủ	
4	Khi nhấn bottom tài khoản	Hiển thị màn hình tài khoản	

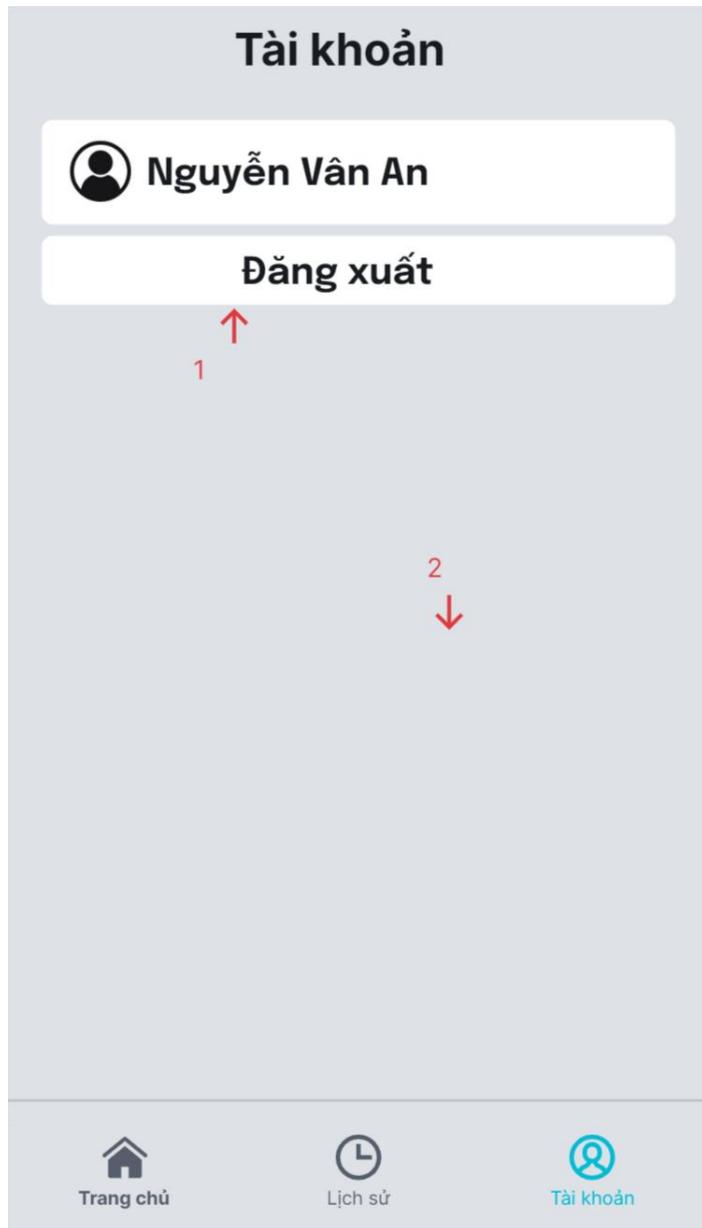
Bảng 27. Biến có lịch sử đặt vé.

Mô tả giao diện lịch sử đặt vé.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	tvTitle	TextView	Tiêu đề giao diện			
2	tvIdBookingSeat	TextView	Mã vé			
	tvRoutes	TextView	Hiển thị tuyến đường			
3	tvreservationTime	TextView	Hiển thị thời gian đặt chỗ			
4	tvPrice	TextView	Hiển thị giá vé			

Bảng 28. Mô tả giao diện lịch sử đặt vé.

4.1.5. Thiết kế giao diện tài khoản.



Hình 58. Thiết kế giao diện tài khoản.

Bảng biến cõ

STT	Điều kiện kích hoạt	Xử lý	Ghi chú
1	Khi nhấn nút đăng xuất	Gọi hàm đăng xuất và quay về trang đăng nhập	
2	Khởi động màn hình	Hiển thị màn hình lịch sử và đọc dữ liệu từ database	

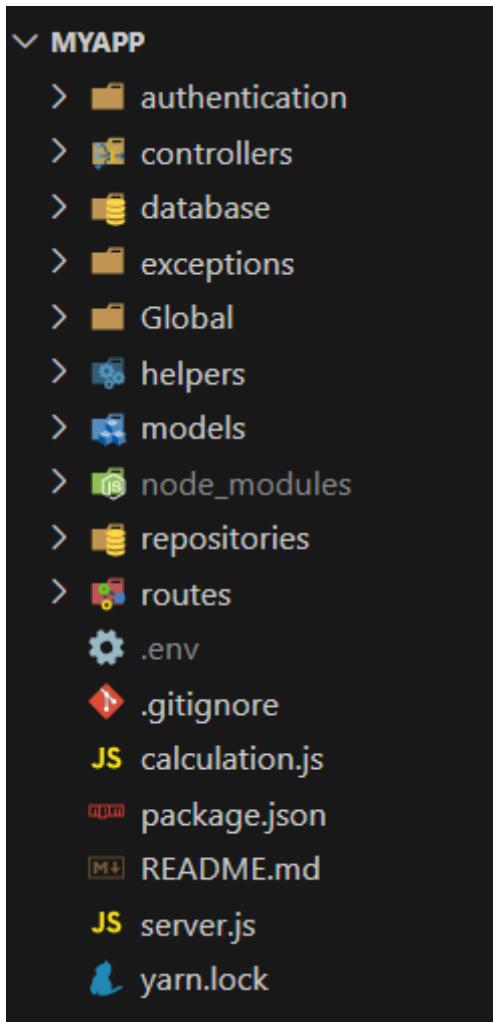
Bảng 29. Biểu đồ giao diện tài khoản.

Mô tả giao diện tài khoản.

Số TT	Tên	Kiểu	Ý nghĩa	Miền giá trị	Miền giá trị mặc định	Ghi chú
1	tvTitle	TextView	Tiêu đề giao diện			
2	icon	imageView	Thể hiện tài khoản			
3	TvName	TextView	Hiển thị tên			
4	Logout	Button	Đăng xuất			

Bảng 30. mô tả giao diện tài khoản.

Chương 5: Viết API và deploy lên server.



Hình 59. Cấu trúc các thư mục viết API.

5.1. Cài đặt và tạo dự án NodeJS trên Windows 11.

Bước 1: Vào trang chủ NodeJS và download về và cài đặt: <https://nodejs.org/en>.

Cài đặt thư viện yarn: npm install – g yarn. Yarn là một công cụ quản lý gói (package manager) phổ biến được sử dụng trong phát triển phần mềm. Yarn thường được sử dụng để quản lý các gói thư viện và phụ thuộc vào các dự án phát triển phần mềm.

Bước 2: Tạo project.

Tạo thư mục mới với tên myapp: mkdir myapp

Tạo project: npm init -y. name: NodeJs. Author: Dang Huynh Nhu Y

Sau khi tạo có file package.json để quản lý các thư viện.

Dùng visual studio code để thực hiện dự án. Tạo một file server.js. Để chạy server.js gõ lệnh trên Terminal: node server.js

Cài thư viện express: yarn add express. Nó là một trong những framework phát triển ứng dụng web phía server phổ biến và mạnh mẽ trong cộng đồng Node.js.

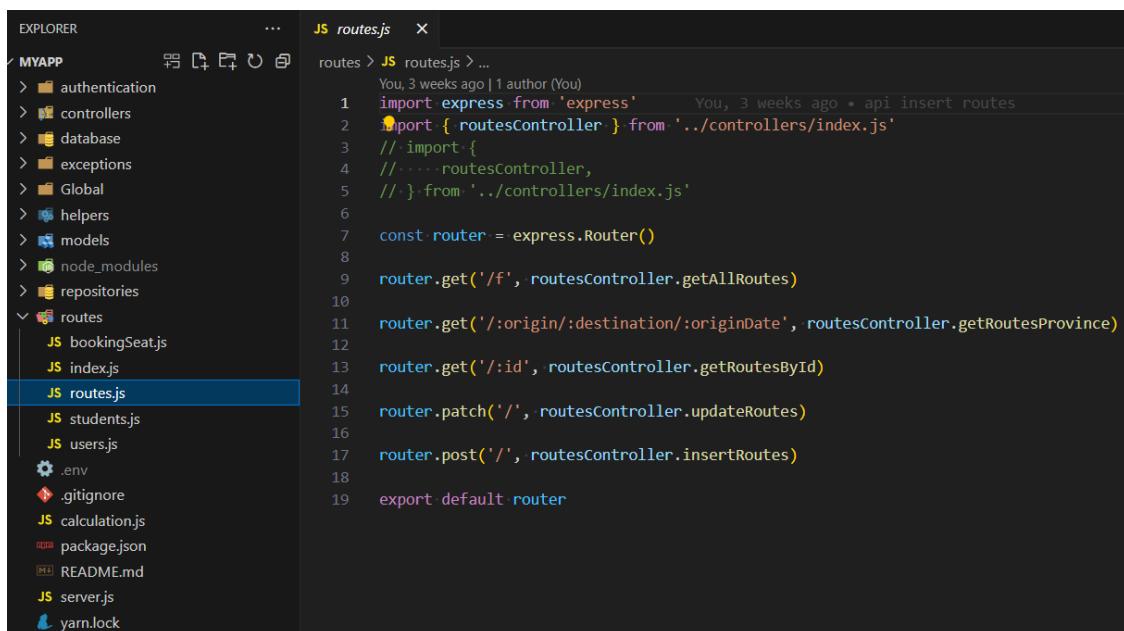
Tạo file .env (Enviroment Variables: quản lý database và password). File env không được push lên githup. Để đọc file .env cần cài đặt thư viện dotenv: yarn add dotenv.

Sử dụng nodemon để không phải chạy lại mỗi lần code thay đổi.

npm install -g nodemon.

Bước 3: Tạo thư mục routes.

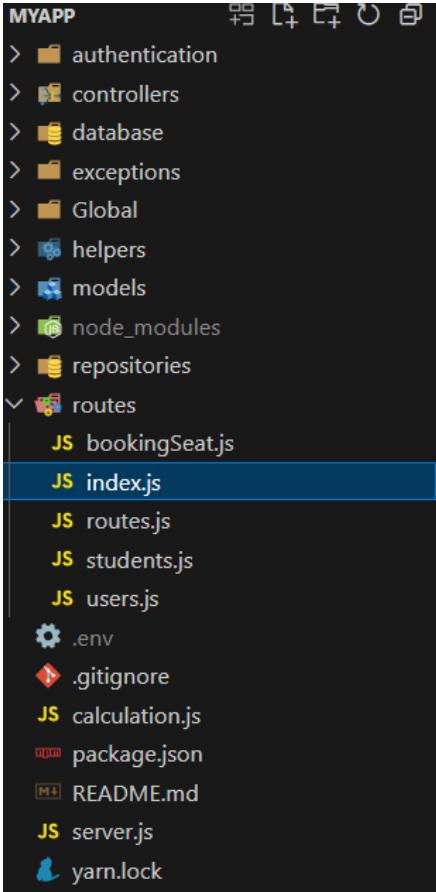
Tạo một thư mục routes để quản lý các file routes. Tạo bao nhiêu thực thi thì tạo bấy nhiêu file routes. Mỗi routes bao gồm các request (get, put, update, delete).



```
You, 3 weeks ago | 1 author (You)
1 import express from 'express' You, 3 weeks ago * api insert routes
2 import { routesController } from '../controllers/index.js'
3 // import {
4 //   routesController,
5 // } from '../controllers/index.js'
6
7 const router = express.Router()
8
9 router.get('/f', routesController.getAllRoutes)
10
11 router.get('/:origin/:destination/:originDate', routesController.getRoutesProvince)
12
13 router.get('/:id', routesController.getRoutesById)
14
15 router.patch('/', routesController.updateRoutes)
16
17 router.post('/', routesController.insertRoutes)
18
19 export default router
```

Hình 60. Tạo thư mục routes.

Tạo file index.js để quản lý các file routes.



```
MYAPP
> authentication
> controllers
> database
> exceptions
> Global
> helpers
> models
> node_modules
> repositories
< routes
  JS bookingSeat.js
  JS index.js
  JS routes.js
  JS students.js
  JS users.js
  .env
  .gitignore
  JS calculation.js
  package.json
  README.md
  JS server.js
  yarn.lock
```

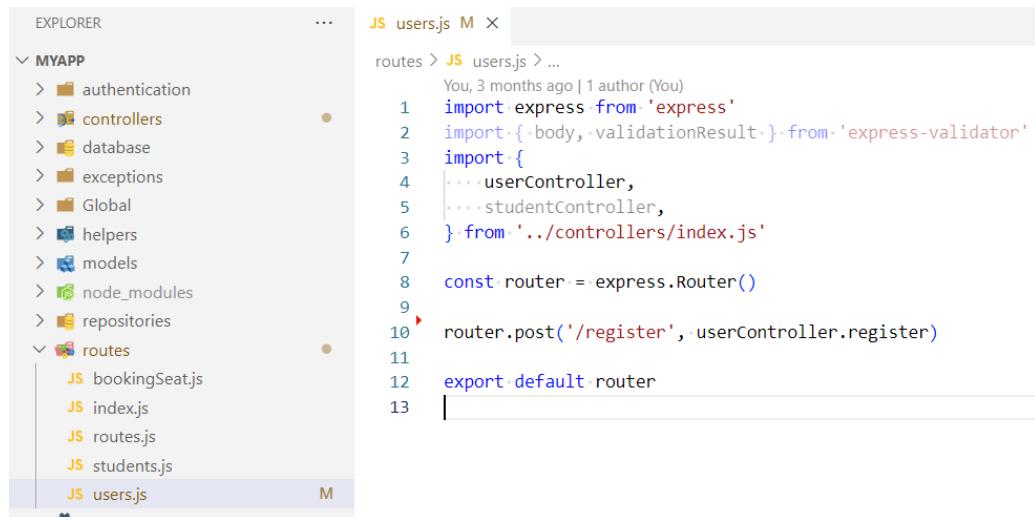
routes > JS index.js

```
You, 3 weeks ago | 1 author (You)
1 import usersRouter from './users.js'
2 import studentRouter from './students.js'
3 import routesRouter from './routes.js'
4 import bookingSeatRouter from './bookingSeat.js'
5
6 export {
7   ...usersRouter,
8   ...studentRouter,
9   ...routesRouter,
10  ...bookingSeatRouter,
11 }
```

Hình 61. Tạo file index.js để quản lý các file routes.

5.2. Viết request register.

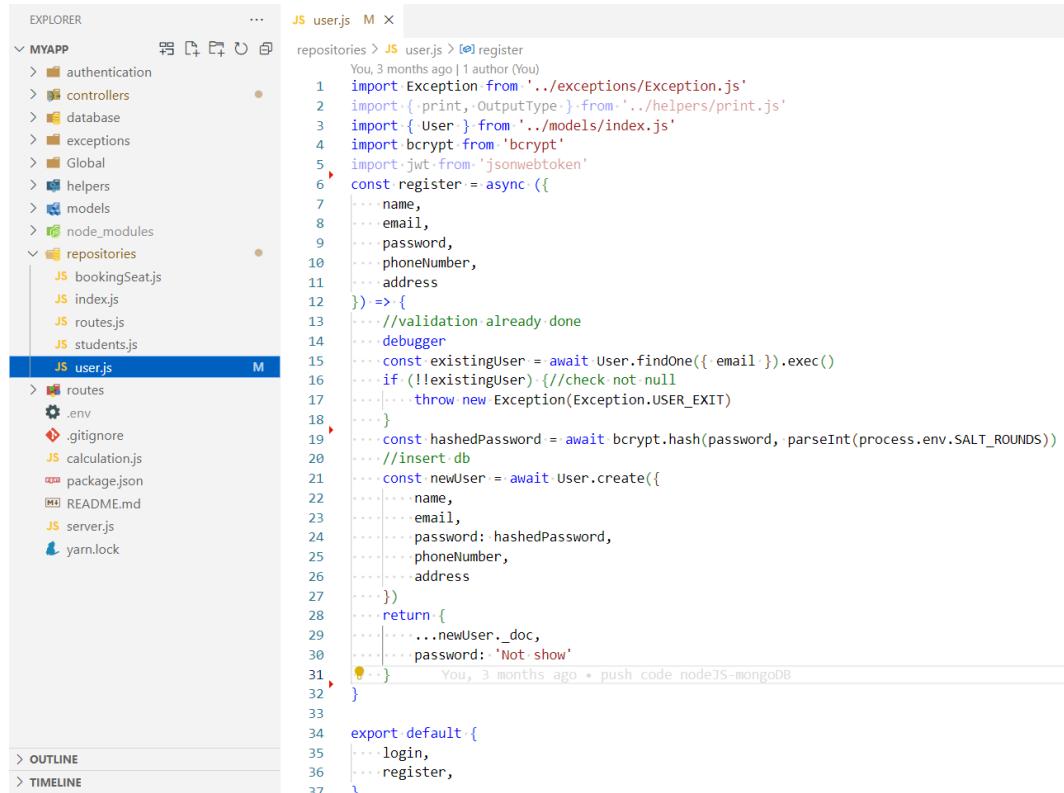
Viết request register trong file user nằm trong thư mục routes.



```
routes > JS users.js > ...
You, 3 months ago | 1 author (You)
1 import express from 'express'
2 import { body, validationResult } from 'express-validator'
3 import {
4     ...userController,
5     ...studentController,
6 } from '../controllers/index.js'
7
8 const router = express.Router()
9
10 router.post('/register', userController.register)
11
12 export default router
13
```

Hình 62. Viết request register trong file user nằm trong thư mục routes.

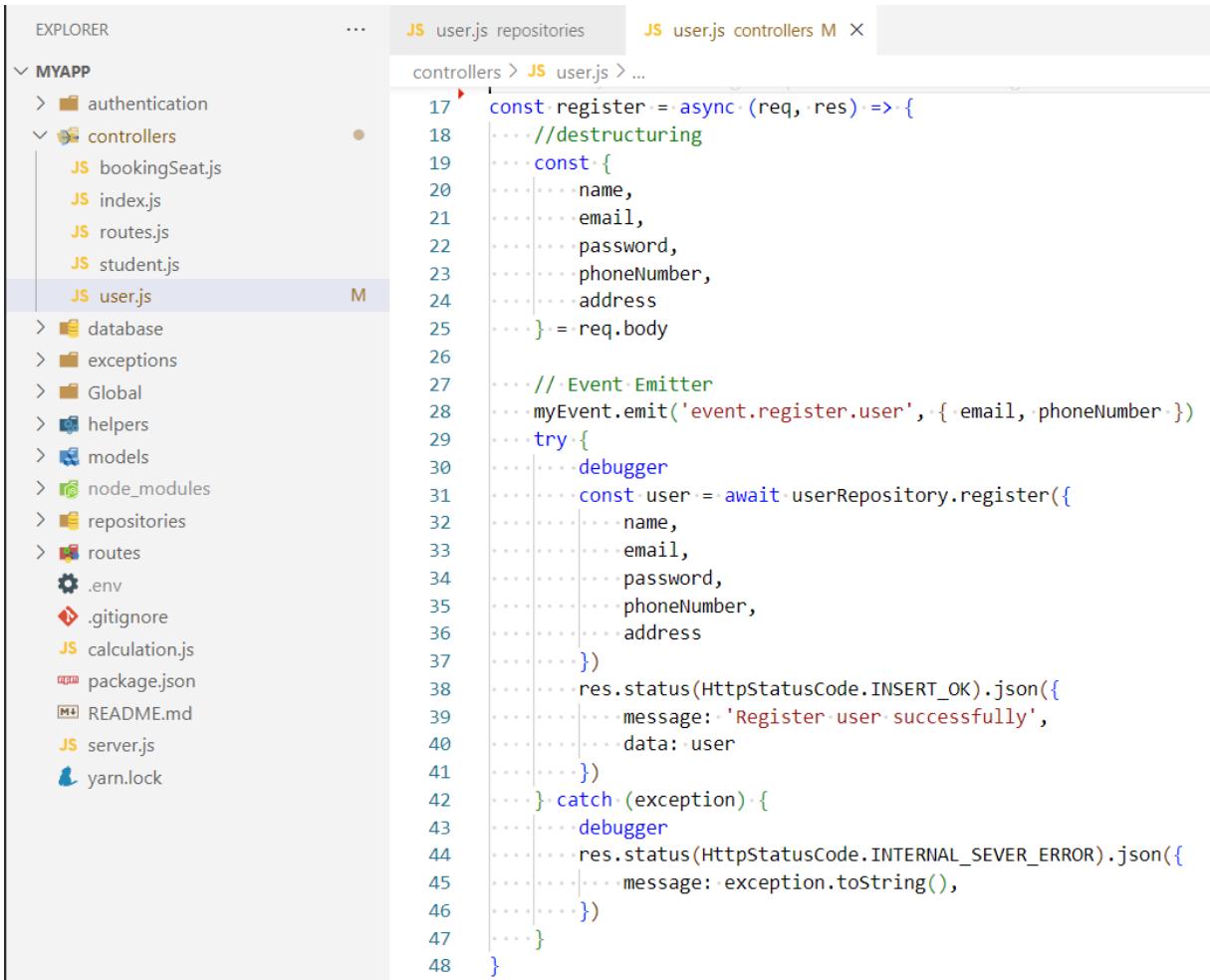
Viết hàm register trong file user.js nằm trong thư mục repositories. Cài thư viện mã hóa password: yarn add bcrypt. Cài thư viện jsonwebtoken: yarn add jsonwebtoken



```
repositories > JS user.js > [o] register
You, 3 months ago | 1 author (You)
1 import Exception from '../exceptions/Exception.js'
2 import { print, OutputType } from '../helpers/print.js'
3 import { User } from '../models/index.js'
4 import bcrypt from 'bcrypt'
5 import jwt from 'jsonwebtoken'
6 const register = async ({ 
7     name,
8     email,
9     password,
10    phoneNumber,
11    address
12 }) => {
13     //validation already done
14     debugger
15     const existingUser = await User.findOne({ email }).exec()
16     if (!existingUser) { //check not null
17         throw new Exception(Exception.USER_EXIT)
18     }
19     const hashedPassword = await bcrypt.hash(password, parseInt(process.env.SALT_ROUNDS))
20     //insert db
21     const newUser = await User.create({
22         name,
23         email,
24         password: hashedPassword,
25         phoneNumber,
26         address
27     })
28     return {
29         ...newUser._doc,
30         password: 'Not show'
31     }
32 }
33
34 export default {
35     login,
36     register,
37 }
```

Hình 63. Viết hàm register trong file user.js nằm trong thư mục repositories.

Viết hàm register trong file user.js nằm trong thư mục controller.



The screenshot shows the VS Code interface with the following details:

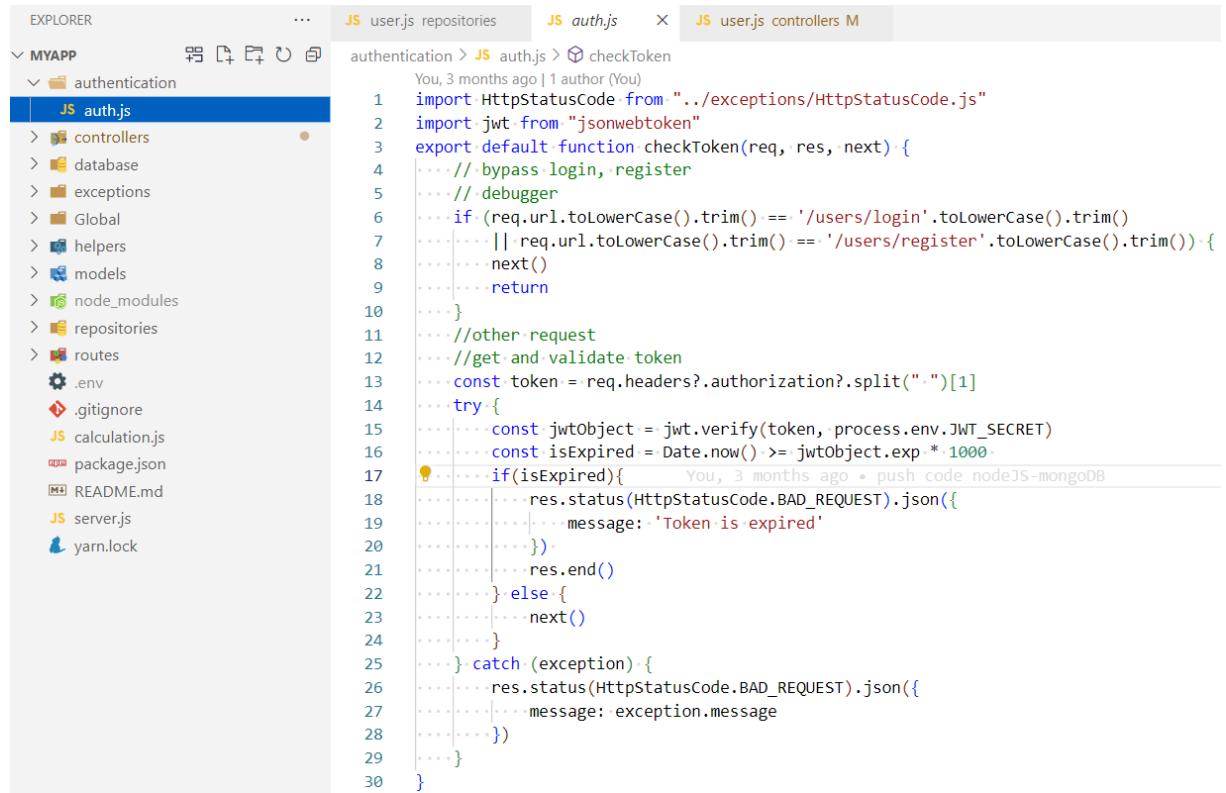
- EXPLORER** sidebar: Shows the project structure under "MYAPP". The "controllers" folder is expanded, showing files: bookingSeat.js, index.js, routes.js, student.js, and user.js (which is currently selected).
- CODE EDITOR**: The active file is "user.js" located at "controllers > JS user.js > ...". The code contains a function named "register".

```
const register = async (req, res) => {
  //destructuring
  const {
    name,
    email,
    password,
    phoneNumber,
    address
  } = req.body

  // Event Emitter
  myEvent.emit('event.register.user', {email, phoneNumber})
  try {
    debugger
    const user = await userRepository.register({
      name,
      email,
      password,
      phoneNumber,
      address
    })
    res.status(HttpStatusCode.INSERT_OK).json({
      message: 'Register user successfully',
      data: user
    })
  } catch (exception) {
    debugger
    res.status(HttpStatusCode.INTERNAL_SEVER_ERROR).json({
      message: exception.toString(),
    })
  }
}
```

Hình 64. Viết hàm register trong file user.js nằm trong thư mục controller.

Tạo một thư mục authentication, tạo file auth.js nằm trong thư mục authentication. Khi các request gửi đến kiểm tra xem có token có chưa và kiểm tra token đúng sẽ cho đi qua. Đối với request login và register thì bỏ qua.



The screenshot shows the VS Code interface with the following details:

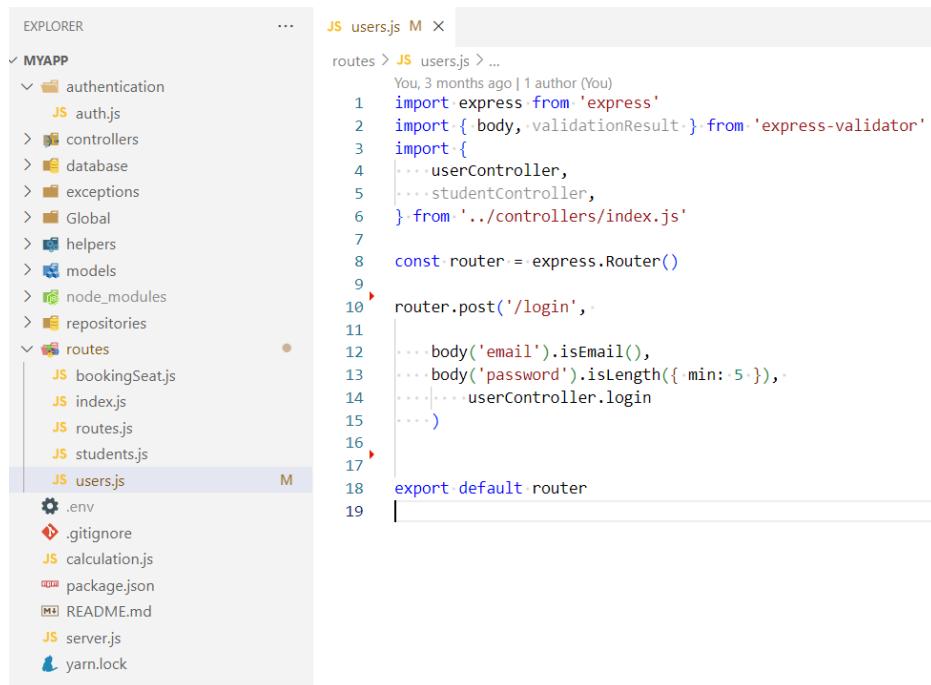
- EXPLORER** sidebar: Shows the project structure under **MYAPP**, including **authentication**, **controllers**, **database**, **exceptions**, **Global**, **helpers**, **models**, **node_modules**, **repositories**, **routes**, **.env**, **.gitignore**, **calculation.js**, **package.json**, **README.md**, **server.js**, and **yarn.lock**.
- CODE EDITOR**: The file **auth.js** is open, showing a function named **checkToken**. The code uses **jsonwebtoken** to verify tokens and handle requests for login and register endpoints.

```
You, 3 months ago | 1 author (You)
1 import HttpStatusCode from "../exceptions(HttpStatusCode.js"
2 import jwt from "jsonwebtoken"
3 export default function checkToken(req, res, next) {
4     // bypass login, register
5     // debugger
6     if (req.url.toLowerCase().trim() === '/users/login'.toLowerCase().trim()
7         || req.url.toLowerCase().trim() === '/users/register'.toLowerCase().trim()) {
8         next()
9         return
10    }
11    // other request
12    // get and validate token
13    const token = req.headers?.authorization?.split(" ")[1]
14    try {
15        const jwtObject = jwt.verify(token, process.env.JWT_SECRET)
16        const isExpired = Date.now() >= jwtObject.exp * 1000
17        if (isExpired) {
18            res.status(HttpStatusCode.BAD_REQUEST).json({
19                message: 'Token is expired'
20            })
21            res.end()
22        } else {
23            next()
24        }
25    } catch (exception) {
26        res.status(HttpStatusCode.BAD_REQUEST).json({
27            message: exception.message
28        })
29    }
30 }
```

Hình 65. Tạo file `auth.js` nằm trong thư mục `authentication`.

5.3. Viết request login.

Viết request login trong file user nằm trong thư mục routes.



```
EXPLORER JS users.js M X
...
routes > JS users.js > ...
You, 3 months ago | 1 author (You)
1 import express from 'express'
2 import { body, validationResult } from 'express-validator'
3 import {
4   ...userController,
5   ...studentController,
6 } from '../controllers/index.js'
7
8 const router = express.Router()
9
10 router.post('/login', [
11   body('email').isEmail(),
12   body('password').isLength({ min: 5 }),
13   ...userController.login
14 ])
15
16
17
18 export default router
19
```

Hình 66. Viết request login trong file user nằm trong thư mục routes.

Cài thư viện express-validator để kiểm tra email và password: yarn add express-validator.

Tổ chức controllers trong dự án. Controllers là nơi nhận các request tương ứng. Tạo thư mục controllers và tạo file user nằm trong controller. Viết hàm login.

```

    controllers > JS user.js > ...
    You, 3 months ago | 1 author (You)
    1 import {
    2   ...body,
    3   ...validationResult
    4 } from 'express-validator'
    5 import {
    6   ...studentRepository,
    7   ...userRepository,
    8 } from '../repositories/index.js'
    9 import { EventEmitter } from 'node:events'
    10 import HttpStatusCode from '../exceptions(HttpStatusCode.js'
    11 const myEvent = new EventEmitter()
    12 //listen
    13 myEvent.on('event.register.user', (params) => {
    14   console.log(`They talked about: ${JSON.stringify(params)}`)
    15 })
    16
    17 const login = async (req, res) => {
    18   const errors = validationResult(req);
    19   if (!errors.isEmpty()) {
    20     return res.status(HttpStatusCode.BAD_REQUEST).json({
    21       errors: errors.array()
    22     })
    23   }
    24   const { email, password } = req.body
    25   //call repository
    26   try {
    27     let existingUser = await userRepository.login({ email, password })
    28     res.status(HttpStatusCode.OK).json({
    29       message: 'Login user successfully',
    30       data: existingUser
    31     });
    32   } catch (exception) {
    33     res.status(HttpStatusCode.INTERNAL_SERVER_ERROR).json({
    34       message: exception.toString(),
    35     })
    36   }
    37 }

```

> OUTLINE
 > TIMELINE

Hình 67. Tạo file user nằm trong controller.

Tạo một file index.js trong controller để quản lý các file controller khác. Các package khác muốn tương tác với controller phải thông qua file index.js.

```

    EXPLORER ... JS index.js < ...
    MYAPP authentication controllers ...
    > controllers ...
    > bookingSeat.js
    JS index.js
    JS routes.js
    JS student.js
    JS user.js
    > database
    > exceptions
    > Global
    M

    controllers > JS index.js ...
    You, 3 weeks ago | 1 author (You)
    1 import userController from './user.js';
    2 import studentController from './student.js';
    3 import routesController from './routes.js';
    4 import bookingSeatController from './bookingSeat.js';
    5
    6 export {
    7   ...userController,
    8   ...studentController,
    9   ...routesController,
    10  ...bookingSeatController
    11 }

```

Hình 68. Tạo một file index.js trong controller để quản lý các file controller khác.

Tổ chức Repositories để lấy dữ liệu dưới Database (MongoDB).
Repositories là nơi lưu trữ và trích xuất dữ liệu.

Tạo file user.js nằm trong thư mục repositories. Viết hàm login.

```

JS user.js      X
repositories > JS user.js > [0] register
You, 3 months ago | 1 author (You)
1 import Exception from '../exceptions/Exception.js'
2 import {print, OutputType} from '../helpers/print.js'
3 import { User } from '../models/index.js'
4 import bcrypt from 'bcrypt'
5 import jwt from 'jsonwebtoken'

6
7 const login = async ({ email, password }) => {
8   //console.log('Login user in user repository', OutputType.INFORMATION)
9   let existingUser = await User.findOne({ email }).exec()
10  if (!existingUser) {
11    // not encrypt password
12    let isMatch = await bcrypt.compare(password, existingUser.password)
13    if (!isMatch) { // isMatch == true
14      // create json web token
15      let token = jwt.sign({
16        data: existingUser
17      },
18      process.env.JWT_SECRET, {
19        // expiresIn: '60', // 1 minute
20        expiresIn: '10 days'
21      }
22      )
23      // clone and add more properties
24      return {
25        ...existingUser.toObject(),
26        password: "not show",
27        token: token
28      }
29    } else {
30      throw new Exception(Exception.WRONG_EMAIL_AND_PASSWORD)
31    }
32  } else {
33    throw new Exception(Exception.WRONG_EMAIL_AND_PASSWORD)
34  }
35 }

```

Hình 69. Tạo file user.js nằm trong thư mục repositories.

Tạo một file index.js để quản lý các file repositories khác.

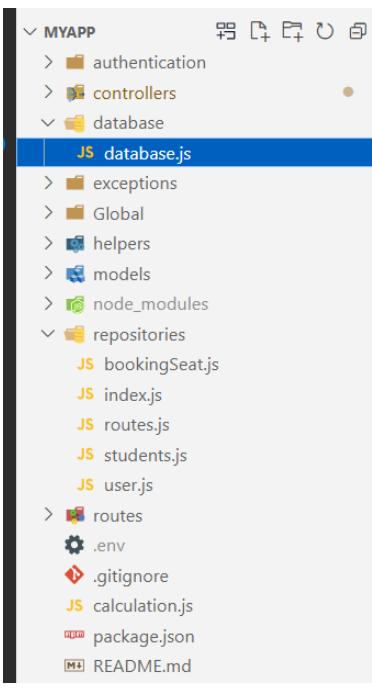
```

JS index.js      X
repositories > JS index.js
You, 3 weeks ago | 1 author (You)
1 import studentRepository from './students.js'
2 import userRepository from './user.js'
3 import routesRepository from './routes.js'
4 import bookingSeatRepository from './bookingSeat.js'
5
6 export {
7   studentRepository,
8   userRepository,
9   routesRepository,
10  bookingSeatRepository,
11 }

```

Hình 70. Tạo một file index.js để quản lý các file repositories khác.

Tạo thư mục database để kết nối với cơ sở dữ liệu. Tạo file database.js



```

database > JS database.js > ...
You, 3 months ago | 1 author (You)
1 import mongoose from "mongoose"; You, 3 months ago • push code no
2 import { print, OutputType } from "../helpers/print.js";
3 import Exception from "../exceptions/Exception.js";
4 mongoose.set('strictQuery', true)
5 async function connect() {
6   try {
7     let connection = await mongoose.connect(process.env.MONGO_URI)
8     print('connect mongoose successful', OutputType.SUCCESS)
9     return connection
10  } catch (error) {
11    //let errorMessage = error.code
12    const { code } = error
13    if (error.code == 8000) {
14      throw new Exception(Exception.WRONG_DB_USERNAME_PASSWORD)
15    } else if (code == 'ENODATA') {
16      ...
17      throw new Exception(Exception.WRONG_DB_CONNECTION_STRING)
18    }
19    throw new Exception(Exception.CANNOT_CONNECT_MONGODB)
20  }
21
22
23 export default connect
24

```

Hình 71. Tạo file database.js để kết nối với cơ sở dữ liệu.

Cài đặt thư viện chalk giúp hiển thị màu sắc trên console: yarn add chalk

Tạo một thư mục helpers, tạo một file print.js để in ra màn hình màu cho lỗi và cảnh báo, thành công.

```

EXPLORER JS print.js JS Exception.js JS HttpStatusCode.js
helpers > JS print.js > ...
1 import chalk from "chalk"; You, 3 months ago | 1 author (You)
2 You, 3 months ago | 1 author (You)
3 class OutputType { You, 3 months ago | 1 author (You)
4   static INFORMATION = "INFORMATION"
5   static SUCCESS = "SUCCESS"
6   static ERROR = "ERROR"
7   static WARNING = "WARNING"//
8 }
9 function print(message, outputType) {
10   switch(outputType) {
11     case OutputType.INFORMATION:
12       console.log(chalk.white(message))
13       break
14     case OutputType.SUCCESS:
15       console.log(chalk.green(message))
16       break
17     case OutputType.WARNING:
18       console.log(chalk.yellow(message))
19       break
20     case OutputType.ERROR:
21       console.log(chalk.red(message))
22       break
23     default:
24       console.log(chalk.white(message))
25   }
26 }
27 export {
28   outputType,
29   print,
30 }

```

Hình 72. Tạo một file print.js để in ra màn hình màu cho lỗi và cảnh báo, thành công.

Tạo một thư mục exceptions trong thư mục tạo file Exceptions.js in ra các lỗi. Tạo file HttpStatusCode.js để trả về các status code khi gửi request.

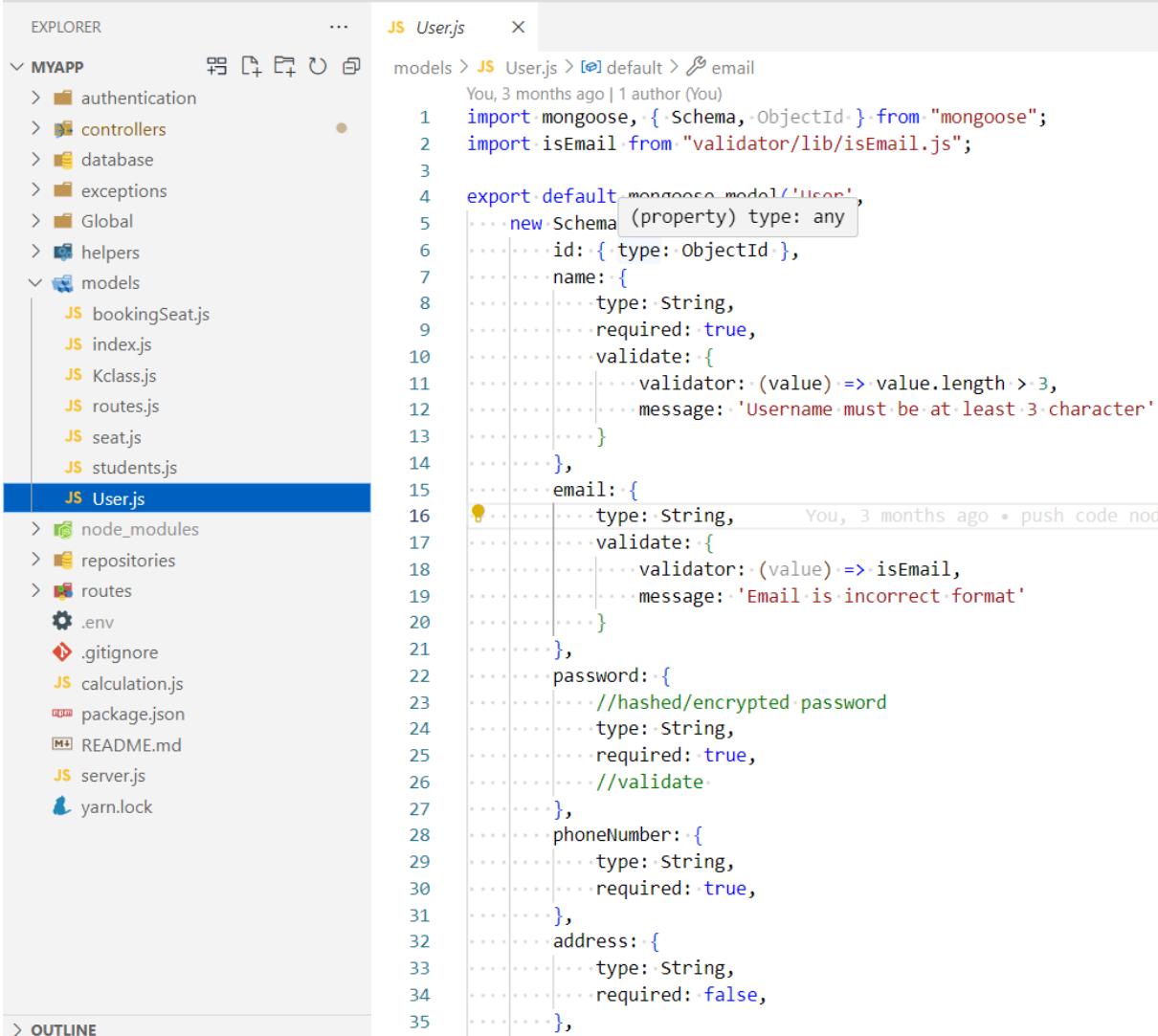
```

EXPLORER JS HttpStatusCode.js X
exceptions > JS HttpStatusCode.js > [?] HttpStatusCode.js
1 const HttpStatusCode = { You, 3 months ago | 1 author (You)
2   OK: 200,
3   INSERT_OK: 201,
4   BAD_REQUEST: 400,
5   NOT_FOUND: 404,
6   INTERNAL_SERVER_ERROR: 500
7 }
8
9 export default HttpStatusCode

```

Hình 73. Tạo file HttpStatusCode.js để trả về các status code khi gửi request.

Tạo thư mục model. Tạo các file model



The screenshot shows the Visual Studio Code interface. The left sidebar (EXPLORER) displays the project structure under 'MYAPP'. The 'models' folder contains several files: bookingSeat.js, index.js, Kclass.js, routes.js, seat.js, students.js, and User.js. The 'User.js' file is currently selected. The right pane (EDITOR) shows the code for 'User.js'. The code defines a mongoose schema for a 'User' model. It includes fields for 'id', 'name' (with a minimum length of 3 characters), 'email' (with validation for length and format), 'password' (with validation for length), 'phoneNumber' (with validation for length), and 'address' (with validation for length). The code uses ES6 syntax and imports from 'mongoose' and 'validator/lib/isEmail.js'.

```
import mongoose, { Schema, ObjectId } from "mongoose";
import isEmail from "validator/lib/isEmail.js";

export default mongoose.model('User',
  new Schema({
    id: { type: ObjectId },
    name: {
      type: String,
      required: true,
      validate: {
        validator: (value) => value.length > 3,
        message: 'Username must be at least 3 character'
      }
    },
    email: {
      type: String,
      validate: {
        validator: (value) => isEmail,
        message: 'Email is incorrect format'
      }
    },
    password: {
      type: String,
      required: true,
      validate: [
        {
          validator: (value) => value.length > 6,
          message: 'Password must be at least 6 character'
        }
      ]
    },
    phoneNumber: {
      type: String,
      required: true,
      validate: [
        {
          validator: (value) => value.length === 10,
          message: 'Phone number must be 10 digits'
        }
      ]
    },
    address: {
      type: String,
      required: false
    }
  })
);
```

Cài thư viện để kiểm tra dữ liệu như email: yarn add validator

5.4. MongoDB và kết nối với NodeJS.

Thực hiện đăng ký và đăng nhập vào mongodb. Vào Data Services và mở Database Access.

The screenshot shows the MongoDB Atlas interface. At the top, there's a navigation bar with the 'Atlas' logo, a dropdown for 'Nguyen's Org...', a gear icon for 'Access Manager', and a 'Billing' link. Below the navigation is a secondary navigation bar with tabs: 'AptechTraining' (selected), a separator, 'Data Services' (highlighted in green), 'App Services', and 'Charts'. On the left, a sidebar has sections for 'DEPLOYMENT' (Database selected), 'Data Lake', 'PREVIEW', 'SERVICES' (Triggers, Data API, Data Federation), and 'SECURITY' (Database Access selected, Network Access, Advanced). The main content area is titled 'NGUYEN'S ORG - 2021-04-01 > APTECHTRAINING' and 'Database Deployments'. It shows a cluster named 'Cluster0' with buttons for 'Connect', 'View Monitoring', 'Browse Collections', and '...'. A callout box says 'Enhance Your Experience' with a message about upgrading for better metrics. It also shows performance metrics: R 0, W 0, Last 6 hours, and 0.2/s. A line graph is visible on the right. A search bar at the top says 'Find a database deployment...'. A 'Find a database deployment...' button is also present.

Chọn ADD NEW DATABASE USER

The screenshot shows the 'Database Access' page under 'Database Users'. It has tabs for 'Database Users' (selected) and 'Custom Roles'. A table lists a single user entry: 'User Name': 'A_hoangnd', 'Authentication Method': 'SCRAM', 'MongoDB Roles': 'atlasAdmin@admin', 'Resources': 'All Resources', and 'Actions' with 'EDIT' and 'DELETE' buttons. A green button '+ ADD NEW DATABASE USER' is located at the top right of the table.

Đặt tên và password, phần Built-in Role chọn Read and write to any database. Chọn add User.

Password Authentication

HIDE

Autogenerate Secure Password
 Copy

Database User Privileges

Configure role based access control by assigning database users a mix of one built-in role, multiple custom roles, and multiple specific privileges. A user will gain access to all actions within the roles assigned to them, not just the actions those roles share in common. You must choose at least one role or privilege. [Learn more about roles.](#)

Built-in Role

Select one [built-in role](#) for this user.

Read and write to any database

1 SELECTED

Custom Roles

Select your [pre-defined custom role\(s\)](#). Create a custom role in the [Custom Roles](#) tab.

Specific Privileges

Select multiple privileges and what database and collection they are associated with. Leaving collection blank will grant this role for all collections in the database.

Restrict Access to Specific Clusters/Federated Database Instances

Enable to specify the resources this user can access. By default, all resources in this project are accessible.

Temporary User

This user is temporary and will be deleted after your specified duration of 6 hours, 1 day, or 1 week.

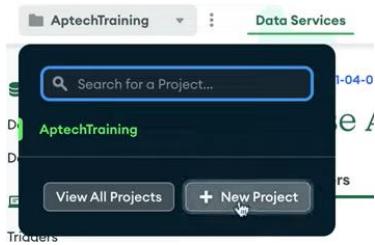
Cancel
 Add User

Sau khi add user

Database Access

User Name	Authentication Method	MongoDB Roles	Resources	Actions
hoangnd	SCRAM	atlasAdmin@admin	All Resources	
tutorial	SCRAM	readWriteAnyDatabase@admin	All Resources	

Tạo một project.



Đặt tên cho project và nhấn Next.

The screenshot shows the 'Create a Project' interface. On the left sidebar, 'Projects' is selected. The main area has a header 'NGUYEN'S ORG - 2021-04-01 > PROJECTS'. Below it, a 'Name Your Project' input field contains 'NodeJSTutorial2023'. To the right are 'Add Members' and two buttons: 'Cancel' and 'Next' (highlighted with a green border).

Chọn Create project

The screenshot shows the 'Add Members and Set Permissions' step. The 'Name Your Project' field now has a green checkmark icon and the text '✓ Name Your Project'. The 'Add Members' button is visible. Below, there's a section for 'Add Members and Set Permissions' with an input field for email addresses and a dropdown for 'Project Owner' set to 'sunlight4d@gmail.com (you)'. At the bottom are 'Cancel', 'Go Back', and a large green 'Create Project' button with a hand cursor icon.

Chọn Build a Database

The screenshot shows the MongoDB Atlas interface under the 'Database Deployments' section. On the left, there's a sidebar with tabs for 'DEPLOYMENT' (selected), 'Database' (selected), 'PREVIEW' (disabled), 'SERVICES' (Triggers, Data API, Data Federation), 'SECURITY' (Database Access, Network Access, Advanced), and 'New On Atlas' (4). A note at the top says 'Current IP Address not added. You will not be able to connect to databases from this address.' To the right, there's a large 'Create a database' button with a green icon, followed by the text 'Choose your cloud provider, region, and specs.' Below it is another 'Build a Database' button.

Chọn Create Clutes

The screenshot shows the MongoDB Atlas cluster creation interface. It starts with a map view of global regions: Montreal (ca-central-1), London (eu-west-2), Seoul (ap-northeast-2), Hong Kong (ap-east-1) (highlighted with a green border), Singapore (ap-southeast-1), Jakarta (ap-southeast-3), Osaka (ap-northeast-3), Bahrain (me-south-1), and Cape Town (af-south-1). Below the map, there are sections for 'Cluster Tier' (M0 Sandbox (Shared RAM, 512 MB Storage) - Encrypted) and 'Additional Settings'. Under 'Additional Settings', there are options for 'Turn on Backup (M2 and up)' (disabled), 'Termination Protection' (NEW, disabled), and a note about preventing accidental deletion. At the bottom, there's a 'FREE' badge, a note about upgrading to a production cluster, a 'Back' link, and a 'Create Cluster' button.

Đặt tên username và password

The screenshot shows the MongoDB Atlas Data Services interface. At the top, there's a navigation bar with tabs for 'NodeJSTutorial...', 'Data Services' (which is underlined in green), 'App Services', and 'Charts'. Below the navigation, there's a section titled 'DEPLOYMENT' with options for 'Database', 'Data Lake' (with a 'PREVIEW' button), and 'Services' (with 'Triggers', 'Data API', and 'Data Federation'). On the left, there's a 'SECURITY' sidebar with 'Quickstart' (which is highlighted in green), 'Database Access', 'Network Access', and 'Advanced'. A 'New On Atlas' button with a '4' notification is also present. The main content area is titled 'How would you like to authenticate your connection?' and offers two options: 'Username and Password' (selected) and 'Certificate'. A detailed description explains that users will be given 'read and write to any database privilege' by default. Below this, there are fields for 'Username' (set to 'tutorial') and 'Password' (set to 'Abc123456789'). There's also an 'Autogenerate Secure Password' button and a 'Create User' button. A 'Copy' button is located at the bottom right of the password field.

Update password

This screenshot shows the same 'Data Services' interface as the previous one, but it's focused on updating a user. The 'Username' field is still 'tutorial' and the 'Password' field is still 'Abc123456789'. The 'Update Password' button is now highlighted in green. The other buttons ('Cancel', 'Autogenerate Secure Password', and 'Copy') are in their original grey state.

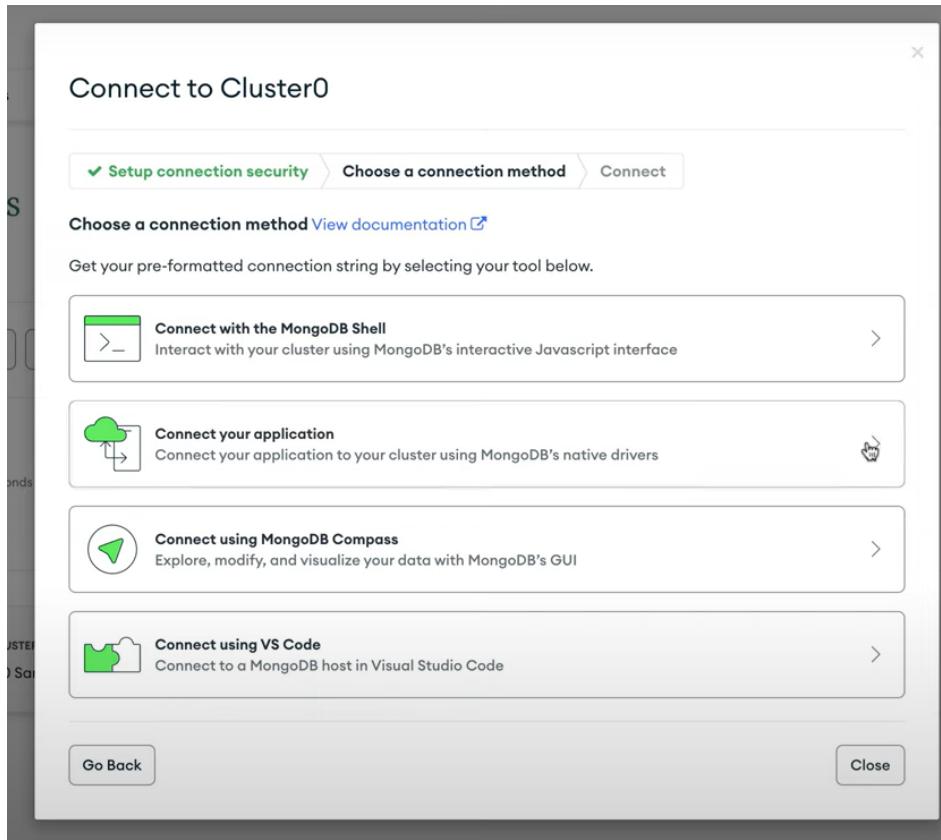
Chọn môi trường My Local Environment, chọn add my current IP Address để cấp quyền truy cập vào database. Sau đó nhán

The screenshot shows the 'IP Access List' configuration page. At the top, there are two options: 'My Local Environment' (selected) and 'Cloud Environment'. Below each option is a brief description and a small icon. The 'My Local Environment' section says: 'Use this to add network IP addresses to the IP Access List. This can be modified at any time.' The 'Cloud Environment' section says: 'Use this to configure network access between Atlas and your cloud or on-premise environment. Specifically, set up IP Access Lists, Network Peering, and Private Endpoints.' A blue 'ADVANCED' button is visible in the top right corner. The main area is titled 'Add entries to your IP Access List'. It contains a note: 'Only an IP address you add to your Access List will be able to connect to your project's clusters. You can manage existing IP entries via the [Network Access Page](#)'. Below this is a table with columns 'IP Address' and 'Description'. There is a row with input fields 'Enter IP Address' (containing '42.114.121.202/32') and 'Enter description' (containing 'My IP Address'). To the right of these fields are 'EDIT' and 'REMOVE' buttons. An 'Add Entry' button is located below the table. At the bottom right of the main area is a green 'Finish and Close' button.

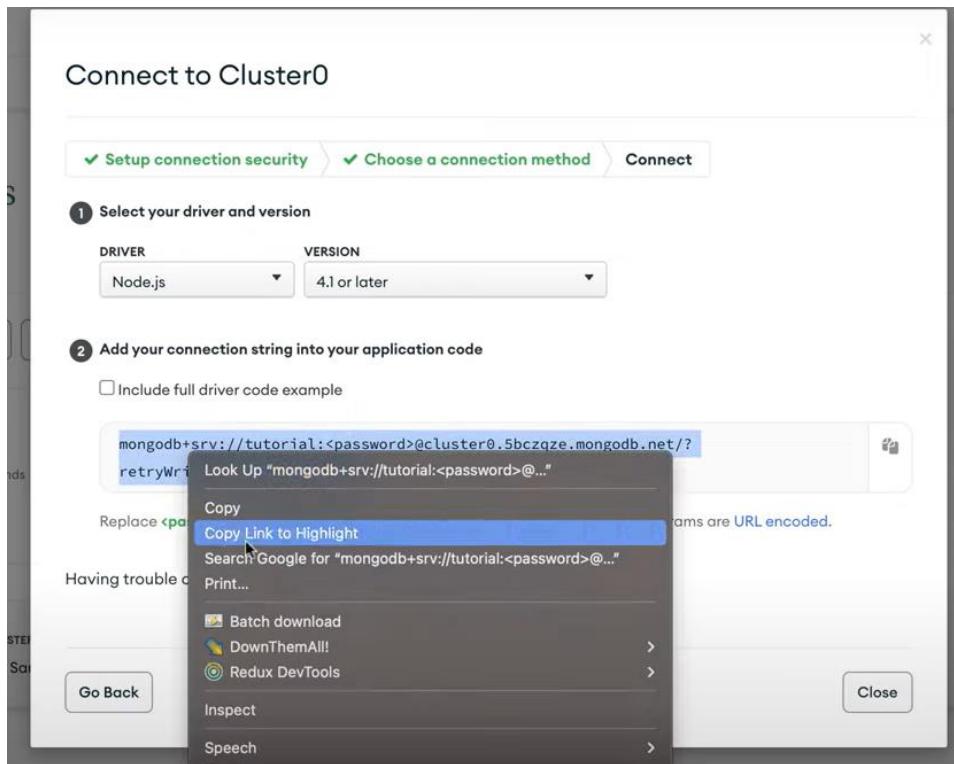
Database Access

Database Users		Custom Roles		
User Name	Authentication Method	MongoDB Roles	Resources	Actions
do_tutorial	SCRAM	readWriteAnyDatabase@admin	All Resources	EDIT DELETE

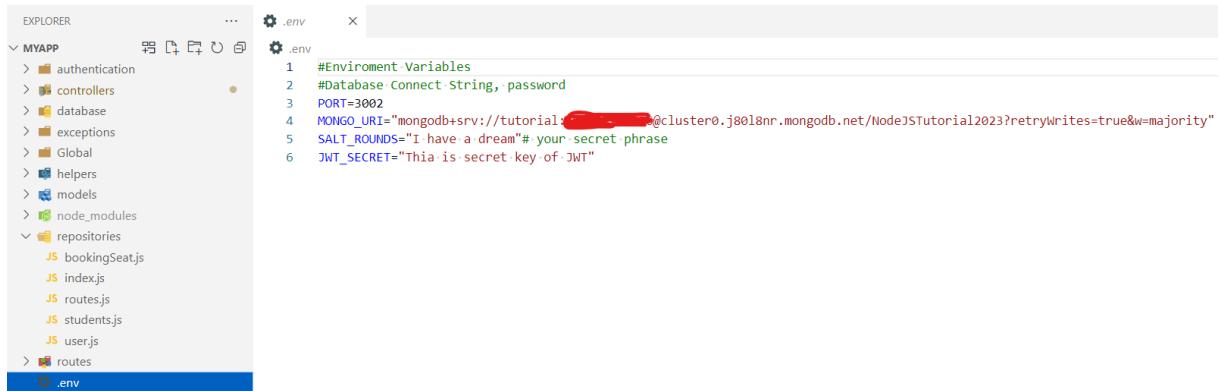
Chọn connect your application



Copy đoạn code mongodb+srv



Dán đoạn code trên vào file .env

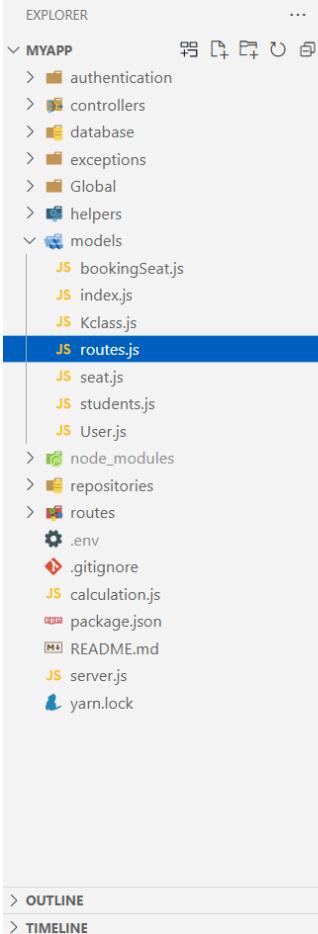


```
#Enviroment Variables
#Database Connect String, password
PORT=3002
MONGO_URL="mongodb+srv://tutorial:[REDACTED]@cluster0.j80l8nr.mongodb.net/NodeJSTutorial2023?retryWrites=true&w=majority"
SALT_ROUNDS="I have a dream" # your secret phrase
JWT_SECRET="This is secret key of JWT"
```

Cài đặt thư viện mongoose để kết nối với cơ sở dữ liệu mongodb: yarn add mongoose.

5.5. Viết request get routes province.

Viết model routes.js nằm trong thư mục model

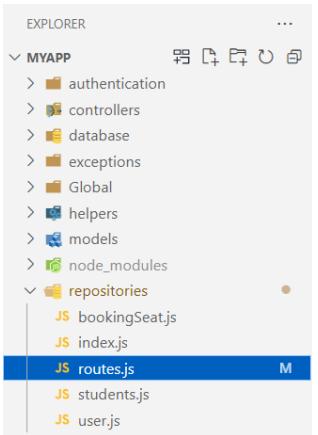


The screenshot shows the VS Code interface with the Explorer sidebar open. The 'MYAPP' folder is expanded, showing subfolders like 'authentication', 'controllers', 'database', 'exceptions', 'Global', 'helpers', 'models', 'node_modules', 'repositories', and files like 'index.js', 'Klass.js', 'routes.js', 'seats.js', 'students.js', and 'User.js'. The file 'routes.js' is selected and highlighted with a blue background. The right-hand editor pane displays the code for 'routes.js'.

```
You, 3 weeks ago | 1 author (You)
1 import { ObjectId, Schema } from "mongoose"
2 import mongoose from "mongoose"
3
4 // Seat Schema
5 const seatSchema = new Schema({
6   ...id: { type: String },
7   ...statusSeat: { type: Boolean },
8 });
9
10 // Trip Schema
11 const tripSchema = new Schema({
12   ...id: { type: String },
13   ...originTime: { type: String },
14   ...destinationTime: { type: String },
15   ...originDate: { type: String },
16   ...destinationDate: { type: String },
17   ...availableSeats: { type: Number },
18   ...seats: [seatSchema]
19 });
20
21 // Province Schema
22 const provinceSchema = new Schema({
23   ...id: { type: String },
24   ...nameProvinces: { type: String },
25   ...locations: [{ type: String }],
26 });
27
28 // Carrier Schema
29 const carrierSchema = new Schema({
30   ...id: { type: String },
31   ...name: { type: String },
32   ...info: [{ type: String }],
33 });
34
35 // Route Schema
36 const routeSchema = new Schema({
37   ...id: { type: ObjectID }
```

Hình 74. Viết model routes.js.

Viết hàm get Routes Province trong file routes.js nằm trong thư mục repositories

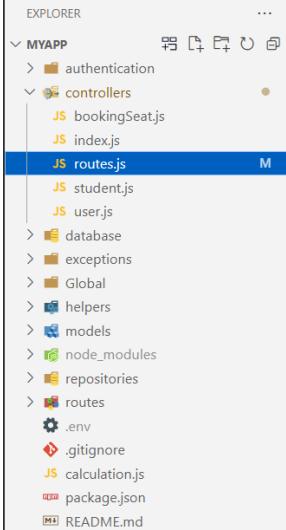


The screenshot shows the VS Code interface with the Explorer sidebar open. The 'MYAPP' folder is expanded, showing subfolders like 'authentication', 'controllers', 'database', 'exceptions', 'Global', 'helpers', 'models', 'node_modules', 'repositories', and files like 'index.js', 'routes.js', 'seats.js', 'students.js', and 'User.js'. The file 'routes.js' is selected and highlighted with a blue background. The right-hand editor pane displays the code for 'routes.js'.

```
You, 9 hours ago | 1 author (You)
1 import { Route } from "../models/index.js" You, 4 weeks ago * api insert routes
2
3 const getRoutesProvince = async (origin, destination, originDate) => { // Thêm hàm này
4   let filteredRoutes = await Route.aggregate([
5     {
6       $match: {
7         "origin.nameProvinces": origin, // Lọc theo tên tỉnh xuất phát
8         "destination.nameProvinces": destination, // Lọc theo tên tỉnh đến
9         "trips.originDate": originDate // Lọc theo thời gian xuất phát
10      }
11    }
12  ]
13 )
14 return filteredRoutes
15 }
```

Hình 75. Viết hàm get Routes Province trong file routes.js nằm trong thư mục repositories.

Viết hàm get routes province nằm trong file routes.js nằm trong thư mục controllers



```

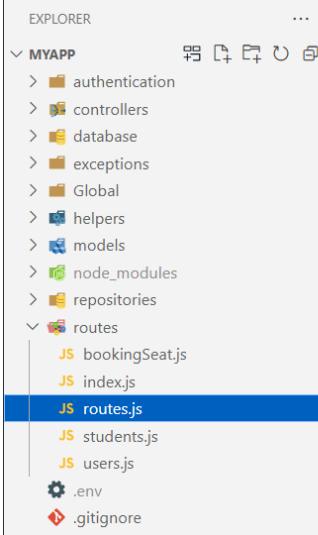
EXPLORER ... JS routes.js M ...
MYAPP ...
  > authentication
  > controllers
    JS bookingSeat.js
    JS index.js
  JS routes.js M
    JS student.js
    JS user.js
  > database
  > exceptions
  > Global
  > helpers
  > models
  > node_modules
  > repositories
  > routes
    .env
    .gitignore
    JS calculation.js
    package.json
  README.md

JS routes.js M ...
controllers > JS routes.js > ...
You, 9 hours ago | 1 author (You)
1 import { body, validationResult } from 'express-validator'
2 import { MAX_RECORDS } from '../Global/constants.js'
3 import HttpStatusCode from '../exceptions(HttpStatusCode.js'
4 import { routesRepository } from '../repositories/index.js'
You, 4 weeks ago * api insert routes
5
6 async function getRoutesProvince(req, res) { //Thêm hàm này
7   try {
8     let origin = req.params.origin // Lấy tham số origin từ req.params
9     let destination = req.params.destination // Lấy tham số destination từ req.params
10    let originDate = req.params.originDate // Lấy tham số originDate từ req.params
11
12    let filteredRoutes = await routesRepository.getRoutesProvince(origin, destination, originDate)
13    res.status(HttpStatusCode.OK).json({
14      message: 'get routes by params successfully',
15      data: filteredRoutes,
16    })
17  } catch (exception) {
18    res.status(HttpStatusCode.INTERNAL_SERVER_ERROR).json({
19      message: exception.message,
20    })
21  }
22 }

```

Hình 76. Viết hàm get routes province nằm trong file routes.js nằm trong thư mục controllers.

Viết request get routes province trong file routes.js nằm trong thư mục routes



```

EXPLORER ... JS routes.js X ...
MYAPP ...
  > authentication
  > controllers
  > database
  > exceptions
  > Global
  > helpers
  > models
  > node_modules
  > repositories
  > routes
    JS bookingSeat.js
    JS index.js
  JS routes.js
    JS students.js
    JS users.js
    .env
    .gitignore

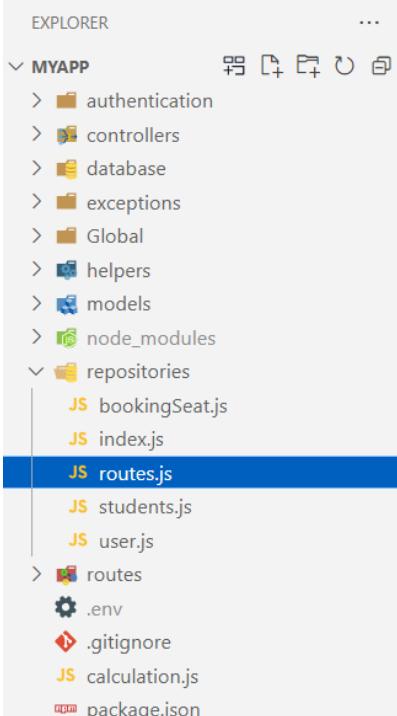
routes > JS routes.js > ...
You, 4 weeks ago | 1 author (You)
1 import express from 'express'
2 import { routesController } from '../controllers/index.js'
3 // import {
4 //   routesController,
5 // } from '../controllers/index.js'
6
7 const router = express.Router()
8
9 router.get('/f', routesController.getAllRoutes)
10
11 router.get('/:origin/:destination/:originDate', routesController.getRoutesProvince)
12
13 router.get('/:id', routesController.getRoutesById)
14
15 router.patch('/', routesController.updateRoutes) You, 4 weeks ago * api insert
16
17 router.post('/', routesController.insertRoutes)
18
19 export default router

```

Hình 77. Viết request get routes province trong file routes.js nằm trong thư mục routes.

5.6. Viết request update routes.

Viết hàm update trong file routes.js nằm trong thư mục repositories.

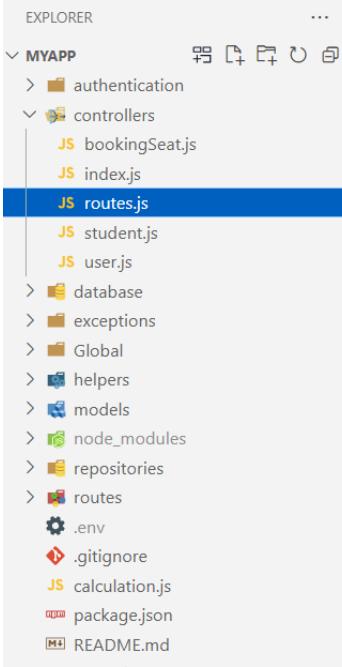


The screenshot shows the VS Code interface with the Explorer sidebar on the left and the code editor on the right. The Explorer sidebar shows a project structure under 'MYAPP' with several folders like authentication, controllers, database, exceptions, Global, helpers, models, node_modules, and repositories. Inside 'repositories', there are files bookingSeat.js, index.js, and routes.js. The routes.js file is selected in the Explorer and is also the active tab in the code editor. The code editor displays the following JavaScript code:

```
const updateRoutes = async ({ id, origin, destination, distance, duration, price, trips, carriers }) => {
  const route = await Route.findById(id)
  route.origin = origin ?? route.origin
  route.destination = destination ?? route.destination
  route.distance = distance ?? route.distance
  route.duration = duration ?? route.duration
  route.price = price ?? route.price
  route.trips = trips ?? route.trips
  route.carriers = carriers ?? route.carriers
  await route.save()
  return route;
};
```

Hình 78. Viết hàm update trong file routes.js nằm trong thư mục repositories.

Viết hàm update routes trong file routes.js nằm trong thư mục controller

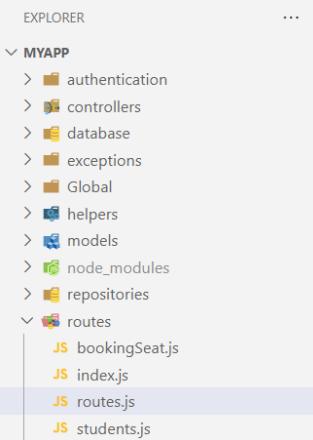


The screenshot shows the VS Code interface with the Explorer sidebar on the left and the code editor on the right. The Explorer sidebar shows a project structure under 'MYAPP' with several files and folders. Inside 'controllers', there are files bookingSeat.js, index.js, and routes.js. The routes.js file is selected in the Explorer and is also the active tab in the code editor. The code editor displays the following JavaScript code:

```
async function updateRoutes(req, res) {
  const { id, origin, destination, distance, duration, price, trips, carriers } = req.body
  try {
    const route = await routesRepository.updateRoutes(req.body)
    res.status(HttpStatusCode.OK).json({
      message: 'update route successfully',
      data: route,
    })
  } catch (exception) {
    res.status(HttpStatusCode.INTERNAL_SERVER_ERROR).json({
      message: exception.message,
    })
  }
}
```

Hình 79. Viết hàm update routes trong file routes.js nằm trong thư mục controller.

Viết request update routes trong file routes.js nằm trong thư mục routes



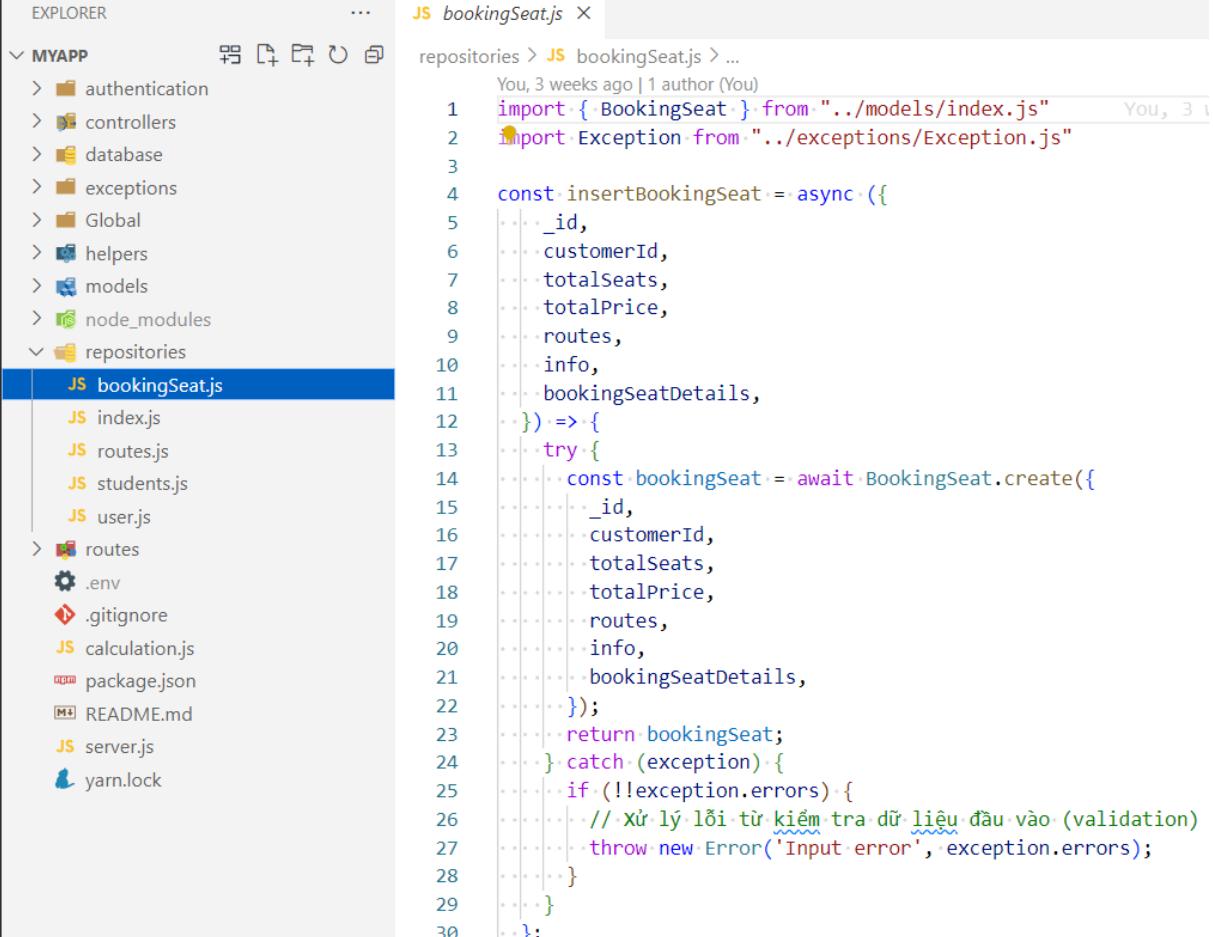
The screenshot shows the VS Code interface. On the left is the Explorer sidebar with a tree view of the project structure under 'MYAPP'. The 'routes' folder is expanded, showing four files: bookingSeat.js, index.js, routes.js (which is selected and highlighted in blue), and students.js. The main editor area on the right displays the content of the 'routes.js' file. The code is written in JavaScript and uses ES6 syntax. It imports 'express' and 'routesController' from other files. It defines a router object and adds several methods: 'get' for '/f', 'get' for '/:origin/:destination/:originDate', 'get' for '/:id', and 'patch' for '/'. The 'patch' method is annotated with a yellow warning icon and a tooltip 'You, 4 weeks ago * api insert rc'. The code is timestamped 'You, 4 weeks ago | 1 author (You)'.

```
routes > JS routes.js > ...
You, 4 weeks ago | 1 author (You)
1 import express from 'express'
2 import { routesController } from '../controllers/index.js'
3 // import {
4 //   routesController,
5 // } from '../controllers/index.js'
6
7 const router = express.Router()
8
9 router.get('/f', routesController.getAllRoutes)
10
11 router.get('/:origin/:destination/:originDate', routesController.getRoutesProvince)
12
13 router.get('/:id', routesController.getRoutesById)
14
15 router.patch('/', routesController.updateRoutes) You, 4 weeks ago * api insert rc
```

Hình 80. Viết request update routes trong file routes.js nằm trong thư mục routes.

5.7. Viết request insert booking seat.

Viết hàm insert booking seat trong file bookingSeat.js nằm trong thư mục repositories.



The screenshot shows the VS Code interface with the Explorer sidebar on the left and the code editor on the right. The Explorer sidebar shows a tree view of the project structure under 'MYAPP'. The 'repositories' folder contains several files: 'index.js', 'routes.js', 'students.js', 'user.js', 'routes', '.env', '.gitignore', 'calculation.js', 'package.json', 'README.md', 'server.js', and 'yarn.lock'. The 'bookingSeat.js' file is selected in the Explorer, and its content is displayed in the code editor. The code is written in JavaScript and defines an asynchronous function 'insertBookingSeat' that creates a new 'BookingSeat' object using the 'create' method from the 'index.js' file. It handles errors and returns the created booking seat or throws an error if validation fails.

```
You, 3 weeks ago | 1 author (You)
1 import { BookingSeat } from '../models/index.js'
2 import Exception from '../exceptions/Exception.js'
3
4 const insertBookingSeat = async ({ 
5   _id,
6   customerId,
7   totalSeats,
8   totalPrice,
9   routes,
10  info,
11  bookingSeatDetails,
12 }) => {
13   try {
14     const bookingSeat = await BookingSeat.create({
15       _id,
16       customerId,
17       totalSeats,
18       totalPrice,
19       routes,
20       info,
21       bookingSeatDetails,
22     });
23     return bookingSeat;
24   } catch (exception) {
25     if (!exception.errors) {
26       // Xử lý lỗi từ kiểm tra dữ liệu đầu vào (validation)
27       throw new Error('Input error', exception.errors);
28     }
29   }
30 };


```

Hình 81. Viết hàm insert booking seat trong file bookingSeat.js nằm trong thư mục repositories.

Viết hàm insert booking seat trong file bookingSeat.js nằm trong thư mục controllers.

```

EXPLORER ... JS bookingSeat.js ×
controllers > JS bookingSeat.js > ...
You, 3 weeks ago | 1 author (You)
1 import {body, validationResult} from 'express-validator' You, 3 weeks ago • A
2 import {MAX_RECORDS} from '../Global/constants.js'
3 import HttpStatusCode from '../exceptions(HttpStatusCode.js'
4 import { bookingSeatRepository } from '../repositories/index.js'
5
6 async function insertBookingSeat(req, res) {
7   try {
8     const bookingSeat = await bookingSeatRepository.insertBookingSeat(req.body)
9     res.status(HttpStatusCode.INSERT_OK).json({
10       message: 'Insert booking seat successfully',
11       data: bookingSeat
12     })
13   } catch (exception) {
14     res.status(HttpStatusCode.INTERNAL_SERVER_ERROR).json({
15       message: 'Cannot insert booking seat: ' + exception,
16       validationErrors: exception.validationErrors
17     })
18   }
}

```

Hình 82. Viết hàm insert booking seat trong file bookingSeat.js nằm trong thư mục controllers.

Viết request insert booking seat trong file bookingSeat.js nằm trong thư mục routes.

```

EXPLORER ... JS bookingSeat.js ×
routes > JS bookingSeat.js > ...
You, 3 weeks ago | 1 author (You)
1 import express from 'express' You, 3 weeks ago • API insert booking seat
2 import { bookingSeatController } from '../controllers/index.js'
3
4 const router = express.Router()
5
6 router.post('/', bookingSeatController.insertBookingSeat)
7 router.get('/:customerId', bookingSeatController.getBookingSeatByCustomerId);
8 router.get('/bookingSeatId/:_id', bookingSeatController.getBookingSeatId);
9
10 export default router

```

Hình 83. Viết request insert booking seat trong file bookingSeat.js nằm trong thư mục routes.

5.8. Viết request get booking seat by id customer.

Viết hàm get booking seat by customer Id trong file bookingSeat.js nằm trong thư mục repositories.



```
repositories > JS bookingSeat.js > [e] getBookingSeatId
27     . . . . . throw new Error('Input error', exception.errors);
28     . . . . .
29     . . . .
30   . . . ;
31
32   const getBookingSeatByCustomerId = async (customerId) => {
33     const bookingSeats = await BookingSeat.find({ customerId: customerId });
34     if (!bookingSeats) {
35       throw new Exception('Cannot find booking seat with id ' + customerId)
36     }
37     return bookingSeats
38 }
```

Hình 84. Viết hàm get booking seat by customer Id trong file bookingSeat.js nằm trong thư mục repositories.

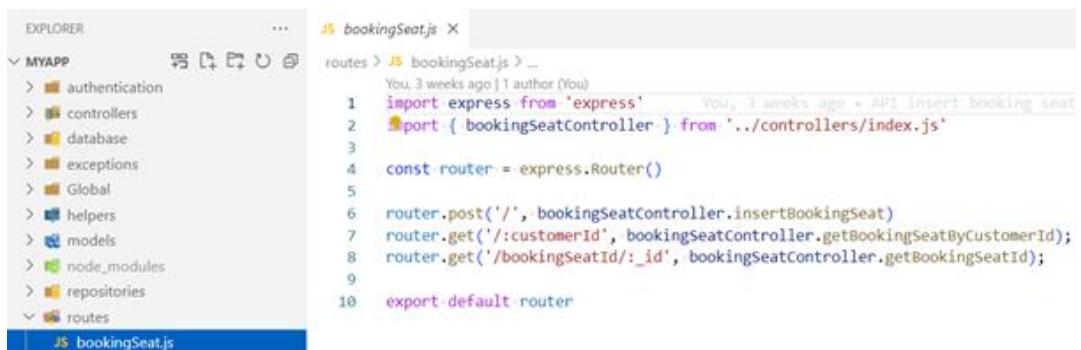
Viết hàm get booking seat by customer id trong file booking.js nằm trong thư mục controller.



```
controllers > JS bookingSeat.js > ...
20
21   async function getBookingSeatByCustomerId(req, res) {
22     const customerId = req.params.customerId
23     try {
24       const bookingSeats = await bookingSeatRepository.getBookingSeatByCustomerId(customerId)
25       res.status(HttpStatusCode.OK).json({
26         message: 'Get detail booking seat successfully',
27         data: bookingSeats,
28       })
29     } catch (exception) {
30       res.status(HttpStatusCode.INTERNAL_SERVER_ERROR).json({
31         message: exception.message,
32       })
33     }
34 }
```

Hình 85. Viết hàm get booking seat by customer id trong file booking.js nằm trong thư mục controller.

Viết request get booking seat by customer id trong routes

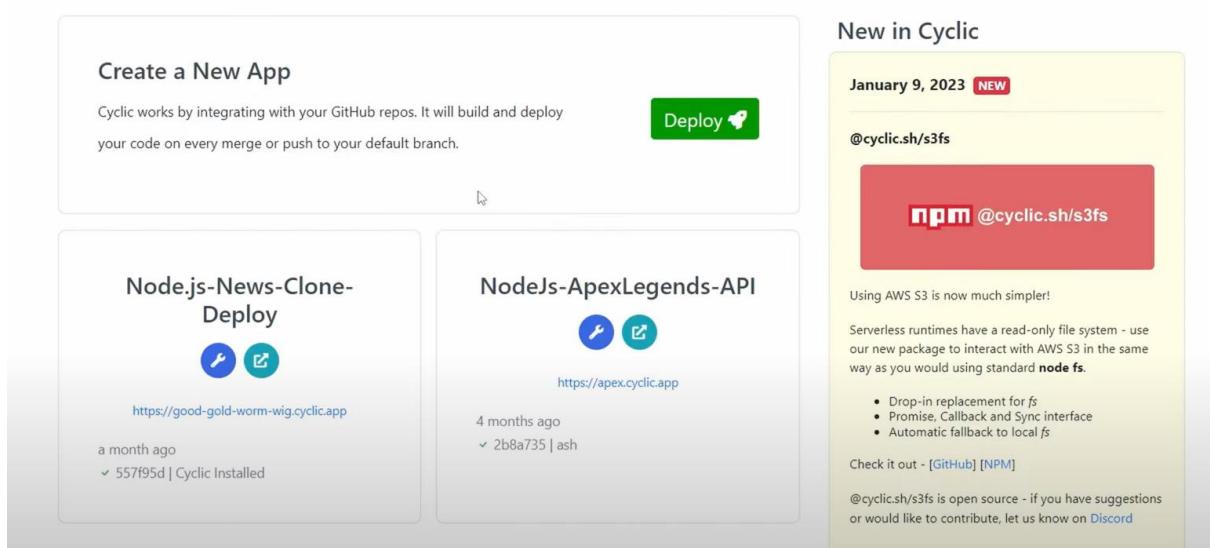


```
routes > JS bookingSeat.js > ...
1 import express from 'express' You, 3 weeks ago | 1 author (You)
2   . . . . . port = { bookingSeatController } from '../controllers/index.js' You, 3 weeks ago * API insert booking seat
3
4 const router = express.Router()
5
6 router.post('/', bookingSeatController.insertBookingSeat)
7 router.get('/:customerId', bookingSeatController.getBookingSeatByCustomerId);
8 router.get('/bookingSeatId/:_id', bookingSeatController.getBookingSeatId);
9
10 export default router
```

Hình 86. Viết request get booking seat by customer id trong routes.

5.9. Deploy API lên Cyclic

Vào trang chủ Cyclic chọn đăng ký: <https://www.cyclic.sh/>. Chọn Continue with GitHub để dễ dàng hơn trong việc deploy api. Sau đó chọn đăng nhập.



Nhấn chọn Deploy.

The screenshot shows the Cyclic website. At the top, there's a navigation bar with the Cyclic logo, 'Docs', 'RaddyTheBrand', and a dropdown menu. Below the header, there are tabs for 'Starter Templates' and 'Link Your Own'. The 'Starter Templates' tab is selected. It features three cards: 'Tiny API' (Vanilla NodeJS), 'ExpressJS API' (Express), and 'Database API' (Express | DynamoDB). Each card has a 'Deploy' and 'GitHub' link. The 'ExpressJS API' card also includes a note about GitHub pop-ups.

Nhấn chọn Link your Own

[Starter Templates](#)

[Link Your Own](#)

Deploy now, we wrote the boilerplate for you

Once you select a starter project, you will see a GitHub pop-up asking you to allow Cyclic to access your fork of the repo.

APIs

Minimal boilerplate to get you going building apps, cron tasks, data manipulation, etc.



Tiny API

Vanilla NodeJS

The smallest possible node backend with no dependencies

[Deploy »](#) [GitHub »](#)



ExpressJS API

Express

Minimal Express backend with just a few lines of code

[Deploy »](#) [GitHub »](#)



Database API

Express | DynamoDB

Basic CRUD API using Express and Cyclic DynamoDB

[Deploy »](#) [GitHub »](#)

Tiếp đến tìm kiếm repositories code api đã push trên github

[Starter Templates](#)

[Link Your Own](#)

Connect a repo to Cyclic

Selecting a repo will trigger a prompt from GitHub asking you to install the [Cyclic GitHub app](#).

Once installed on a repo:

- A webhook will tell Cyclic to build and deploy new changes
- Cyclic will be able to post a or status of the deployment to a commit

Search your repositories... I

Notes-NodeJs-CRUD-MongoDB

Node.js-News-Clone-Deploy

starter-express-api

OnePlant-WordPress-FSE-WooCommerce-Theme

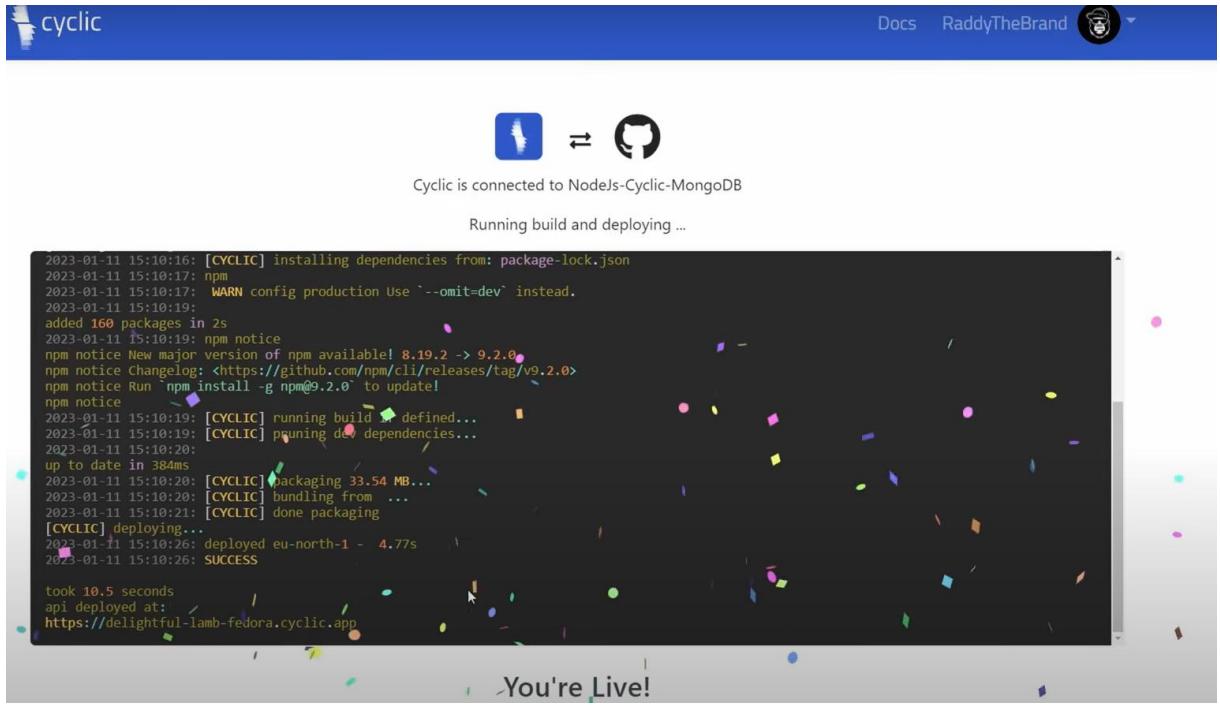
Raddy.co.uk

14.apex-legends-html

Sau đó chọn Connect

The screenshot shows the Cyclic dashboard. At the top, there's a blue header bar with the Cyclic logo, a search bar, and navigation links for 'Docs' and 'RaddyTheBrand'. Below the header, a button says 'Select a project'. A modal window is open, titled 'Connect Cyclic to NodeJs-Cyclic-MongoDB'. It features icons for Cyclic and GitHub, and text explaining that Cyclic will install, build, and start the project. A large 'Connect' button is at the bottom of the modal.

Hiển thị connect thành công



Tiếp đến, quay trở lại và chọn Option and configs

The screenshot shows the 'Create a New App' interface. It includes a summary card for 'NodeJs-Cyclic-MongoDB' and another for 'Node.js-News-Clone-Deploy'. Both cards show deployment details and links to the live apps. To the right, a news feed item for '@cyclic.sh/s3fs' is displayed:

January 9, 2023 NEW

@cyclic.sh/s3fs

npm **@cyclic.sh/s3fs**

Using AWS S3 is now much simpler!

Serverless runtimes have a read-only file system - use our new package to interact with AWS S3 in the same way as you would using standard **node fs**.

- Drop-in replacement for **fs**
- Promise, Callback and Sync interface
- Automatic fallback to local **fs**

Check it out - [\[GitHub\]](#) [\[NPM\]](#)

@cyclic.sh/s3fs is open source - if you have suggestions or would like to contribute, let us know on [Discord](#)

Nhấn nút save để hoàn thành việt cấu hình

Variables

Edit Variables

Environment variables are custom string keys and values (max 255 characters each), set on a per environment basis. The variables are exposed in your code through the `process.env` object.

CYCLIC_URL	https://delightful-lamb-fedora.cyclic.app		Remove
CYCLIC_DB	delightful-lamb-fedoraCyclicDB		Remove
CYCLIC_BUCKET_NAME	cyclic-delightful-lamb-fedora-eu-north-1		Remove
CYCLIC_APP_ID	delightful-lamb-fedora		Remove
MONGO_URI	mongodb+srv://raddy:pnlGiFZVTq6L8hd0@cluster0.k3eethp.mongodb.net/test		Remove

Create New: Variable Name Variable Value **+ Add Variable**

⚠ Changes are applied to app immediately on **Save** **Save**

Đường link API: App URL

NodeJs-Cyclic-MongoDB

prod ➔

https://delightful-lamb-fedora.cyclic.app ↗
https://github.com/RaddyTheBrand/NodeJs-Cyclic-MongoDB ↗

Overview Deployments Logs Transactions Environments Variables Data / Storage Cron Auth Advanced

Get started

1. Give your app a custom url at [Environments](#)
2. See what's happening on the backend with [Logs](#) or [Transactions](#)
3. Configure your `process.env` with [Variables](#)
4. Push a commit and check out [Deployments](#)

Overview

App URL: <https://delightful-lamb-fedora.cyclic.app> ID: [delightful-lamb-fedora](#)

Code Repository: <https://github.com/RaddyTheBrand/NodeJs-Cyclic-MongoDB>

Request Metrics

For requests received by over the most recent 30 days for the selected time period.

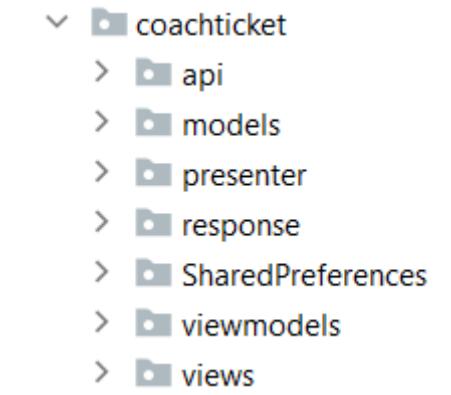
5.10. Link githup API

Link: https://github.com/ydang2002/My_App

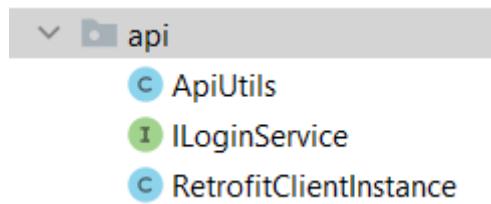
Chương 6: Lập trình và kiểm thử

6.1. Lập trình

6.1.1. Cấu trúc các thư mục

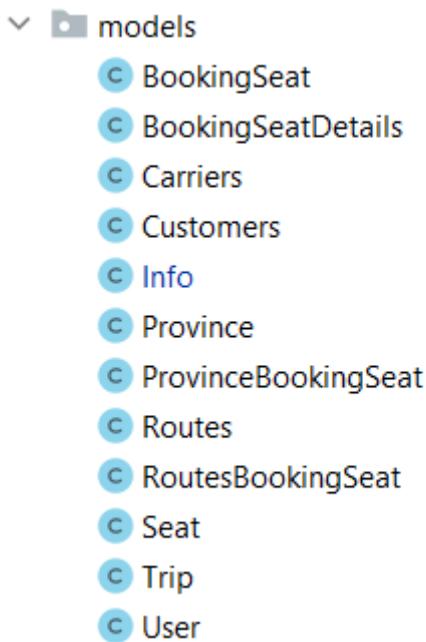


Hình 87. Cấu trúc các thư mục.



Hình 88. Thư mục api.

Thư mục api chứa các file java định nghĩa các class và interface cho API. Các lớp này xác định các phương thức để thực hiện các thao tác trên API. Chẳng hạn như lấy dữ liệu, cập nhật dữ liệu, thêm dữ liệu.



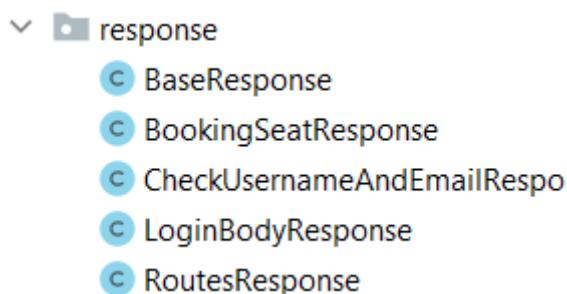
Hình 89. Thư mục models.

Thư mục models chứa các file java định nghĩa các đối tượng, các lớp này định nghĩa các thực thể.



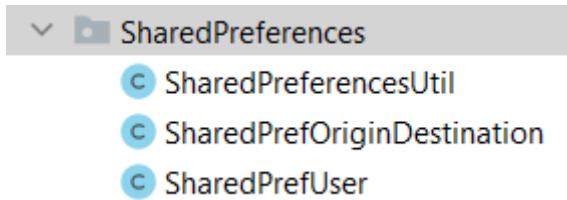
Hình 90. Thư mục presenter.

Thư mục Presenter chứa class chịu trách nhiệm điều hướng các nút trên giao diện.



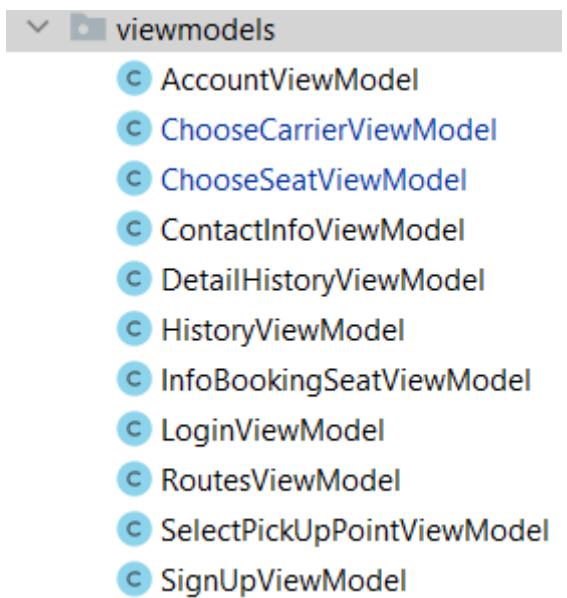
Hình 91. Thư mục response.

Thư mục response thực hiện get api.



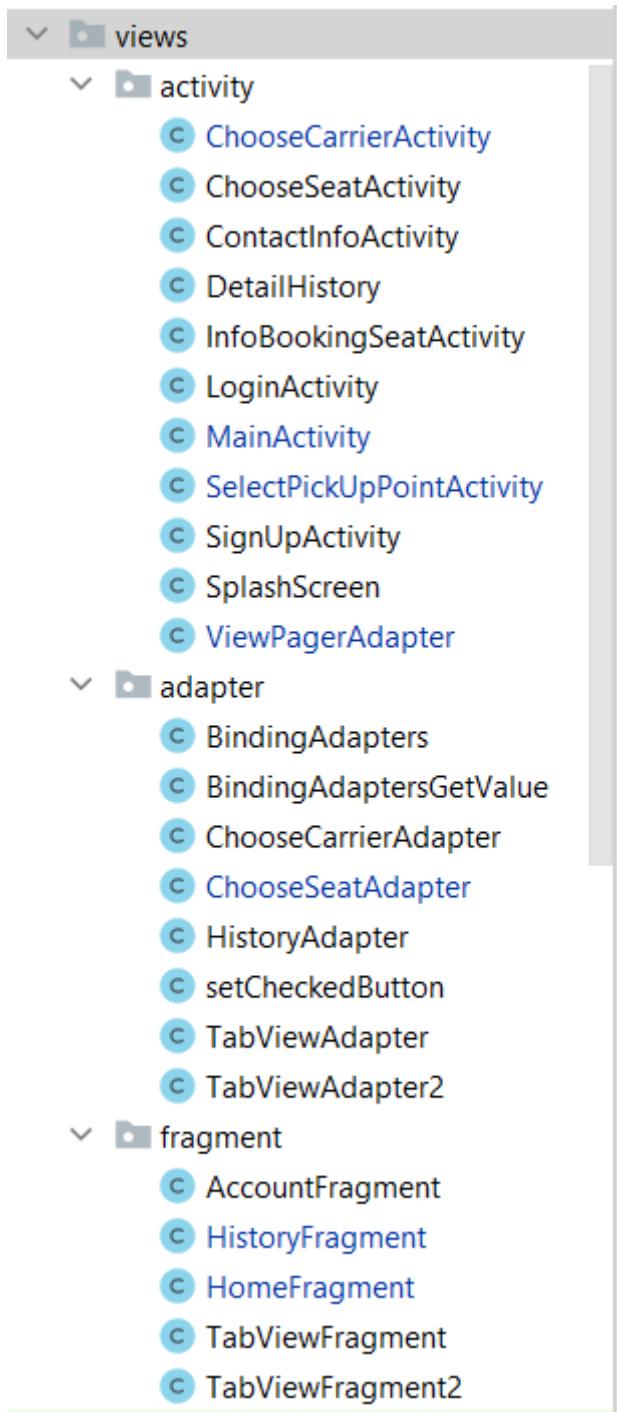
Hình 92. Thư mục Shared Preferences.

Thư mục Shared Preferences chứa các file dùng để lưu trữ dữ liệu dưới dạng cặp khóa-giá trị. Dữ liệu được lưu trữ là dữ liệu cục bộ.



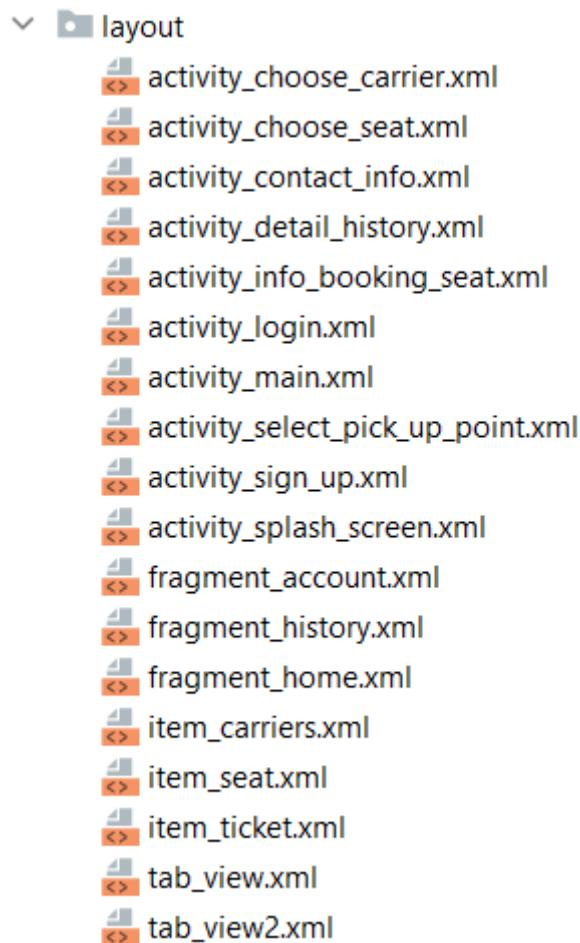
Hình 93. Thư mục viewmodels

Thư mục viewmodel chứa các file java làm lớp trung gian xử lý logic giữa view và model. Viewmodel có trách nhiệm quản lý dữ liệu .



Hình 94. Thư mục view.

Thư mục view chứa các file java Activity, Fragment, Adapter, để xử lý và hiển thị dữ liệu cho người dùng.



Hình 95. Thư mục layout các file xml

Thư mục layout chứa các file xml giao diện.

6.1.2. Màn hình đăng nhập

Code giao diện bằng xml và xử lý chức năng đăng nhập.



Hình 96. Màn hình đăng nhập.

Hàm thực hiện call api, xử lý đăng nhập tài khoản.

```
public void loginViewModel(Context context, String email, String password) {  
    if (TextUtils.isEmpty(email)) {  
        showToast( msg: "Vui lòng nhập email");  
        return;  
    }  
  
    if (!email.matches(emailPattern)) {  
        showToast( msg: "Vui lòng nhập đúng định dạng email");  
        return;  
    }  
  
    if (TextUtils.isEmpty(password)) {  
        showToast( msg: "Vui lòng nhập mật khẩu");  
        return;  
    }  
  
    if (password.length() < 8) {  
        showToast( msg: "Mật khẩu phải lớn hơn 7 kí tự");  
        return;  
    }  
    progressBar.set(View.VISIBLE);  
    LoginBodyResponse loginBody = new LoginBodyResponse(email, password);  
    Call<BaseResponse<User>> call = iLoginService.login(loginBody);  
    call.enqueue(new Callback<BaseResponse<User>>() {
```

```

call.enqueue(new Callback<BaseResponse<User>>() {
    @LAPTOP-38ED4D03\ASUS *
    @Override
    public void onResponse(Call<BaseResponse<User>> call, Response<BaseResponse<User>> response) {
        progressBar.set(View.GONE);
        Log.e( tag: "loginreponse", msg: "đã vào hàm");
        try {
            if (response.isSuccessful()) {
                Intent intent = new Intent(context, MainActivity.class);
                showToast( msg: "Đăng nhập thành công");
                context.startActivity(intent);
                ((Activity) context).finish();
            } else {
                Log.e( tag: "loginreponse2", msg: "Đăng nhập thất bại");
                showToast( msg: "Đăng nhập thất bại");
            }
        }
        catch(Exception e) {
            Log.e( tag: "ExceptionLoginreponse", msg: "error"+ e.getMessage());
        }
    }
    @LAPTOP-38ED4D03\ASUS
    @Override
    public void onFailure(Call<BaseResponse<User>> call, Throwable t) {
        progressBar.set(View.GONE);
        showToast( msg: "Lỗi kết nối" + t.getMessage());
    }
}

```

Hình 97. Hàm đăng nhập.

6.1.3. Màn hình đăng ký.

Code giao diện đăng ký bằng xml và xử lý chức năng đăng ký.



Hình 98. màn hình đăng ký.

Hàm thực hiện call api, xử lý việc đăng ký tài khoản.

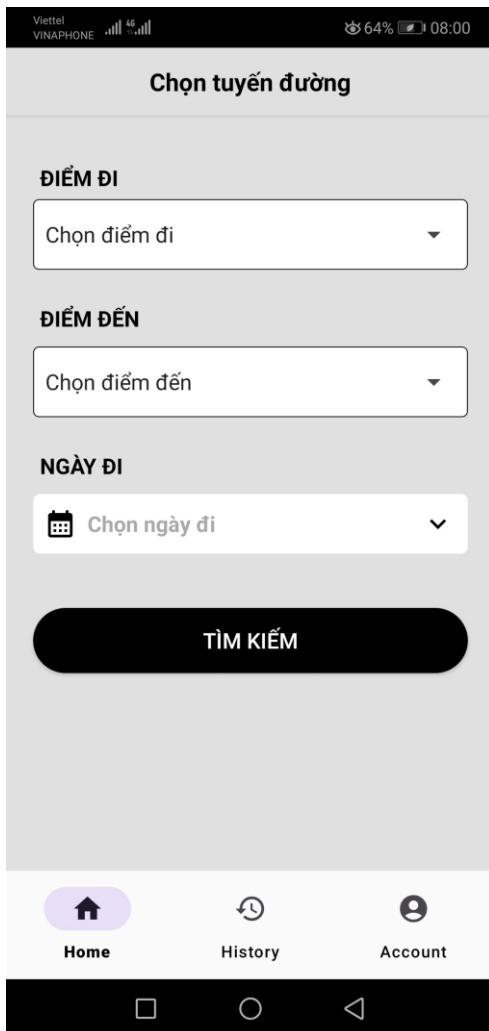
```
public void signUpViewModel(String username, String email, String address, String password, String phone) {
    progressBar.set(View.VISIBLE);
    String token = SharedPreferencesUtil.getToken(context);
    User user = new User(_id: "", email, username, phone, token: "", address, password);
    iLoginService = ApiUtils.getApiService(token);
    // LÀM TÌM HAY LÀM SAO ĐÂU ĐÂU
    iLoginService.register(user).enqueue(new Callback<User>() {
        // LÀM TÌM HAY LÀM SAO ĐÂU ĐÂU
        @Override
        public void onResponse(Call<User> call, Response<User> response) {
            progressBar.set(View.GONE);
            Intent intent = new Intent(context, LoginActivity.class);
            showToast(msg: "Đăng ký thành công");
            context.startActivity(intent);
            ((Activity) context).finish();
        }

        // LÀM TÌM HAY LÀM SAO ĐÂU ĐÂU
        @Override
        public void onFailure(Call<User> call, Throwable t) {
            progressBar.set(View.GONE);
            showToast(msg: "Đăng ký thất bại");
        }
    });
}
```

Hình 99. Hàm đăng ký.

6.1.4. Màn hình chọn tuyến đường.

Code giao diện chọn tuyến đường bằng xml và xử lý chức năng tiềm kiếm tuyến đường.



Hình 100. Màn hình chọn tuyến đường.

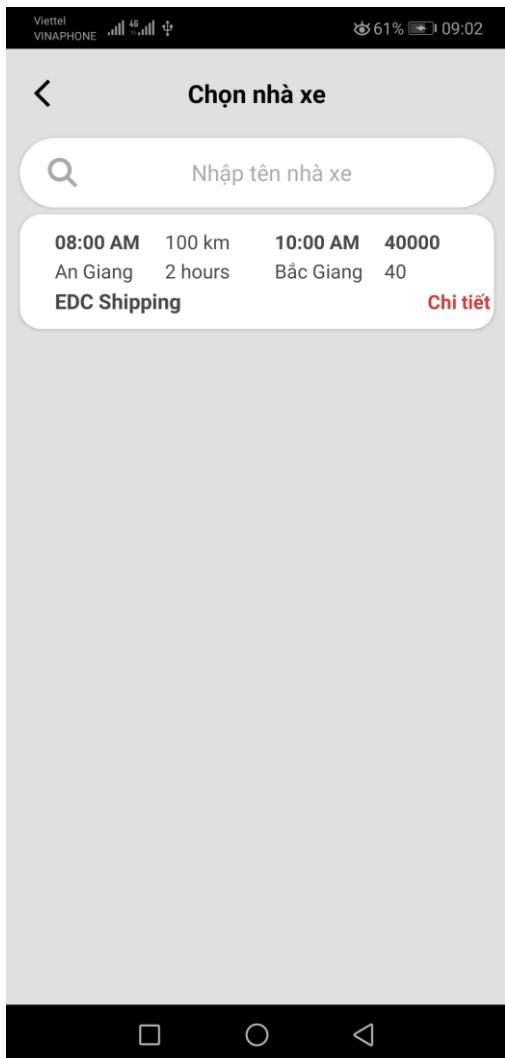
Hàm hiển thị ngày để chọn

```
124     public void onDisplayDatePickerDialogClick() {
125         Calendar calendar = Calendar.getInstance();
126         int year = calendar.get(Calendar.YEAR);
127         int month = calendar.get(Calendar.MONTH);
128
129         int day = calendar.get(Calendar.DAY_OF_MONTH);
130         DatePickerDialog datePickerDialog = new DatePickerDialog(getActivity(),
131             new DatePickerDialog.OnDateSetListener() {
132                 no usages  ↳ LAPTOP-38ED4DO3\ASUS *
133                 @Override
134                 public void onDateSet(DatePicker view, int year, int month, int dayOfMonth) {
135                     date = dayOfMonth + "/" + (month + 1) + "/" + year;
136                     routesViewModel.setDate(dayOfMonth + "-" + (month + 1) + "-" + year);
137                     Log.e( tag: "HienThiNgay: ", date);
138                 }
139             }, year, month, day);
140
141         datePickerDialog.show();
142     }
```

Hình 101. Hàm hiển thị ngày.

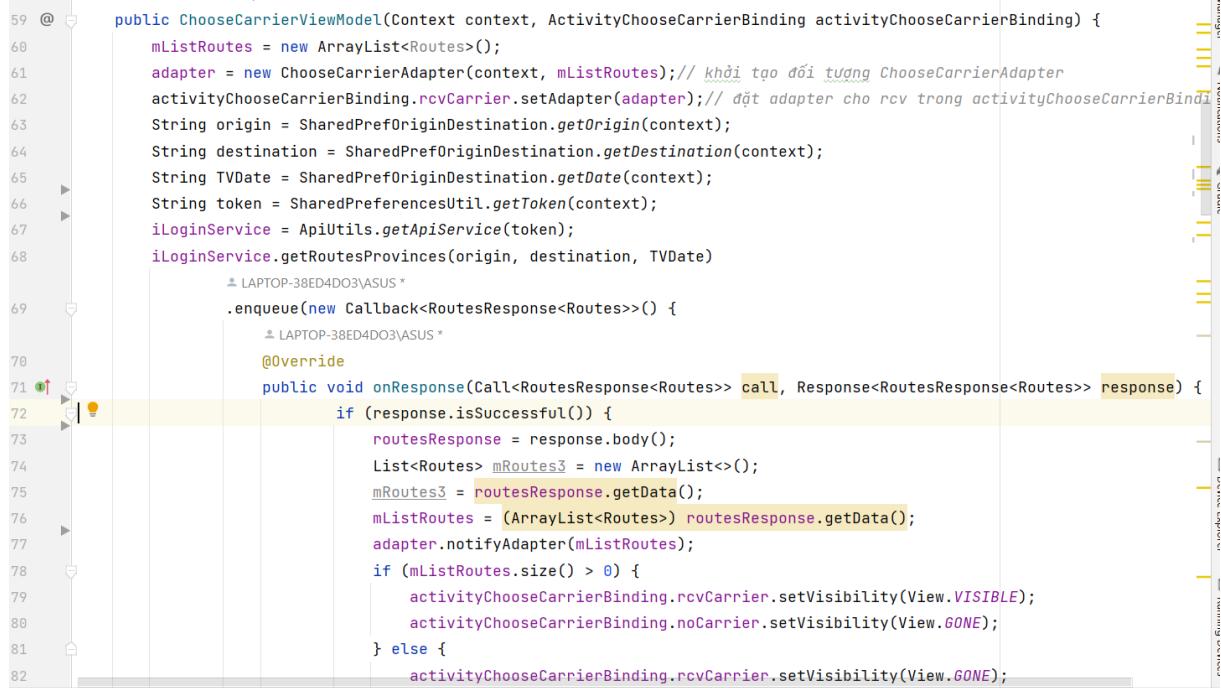
6.1.5. Màn hình chọn nhà xe.

Code giao diện Chọn nhà xe và xử lý chức năng hiển thị các item và tìm kiếm nhà xe.



Hình 102. Màn hình chọn nhà xe.

Hàm thực hiện call api để lấy dữ liệu nhà xe theo ngày đi, điểm đi, điểm đến và đổ vào ChooseCarrierAdapter.

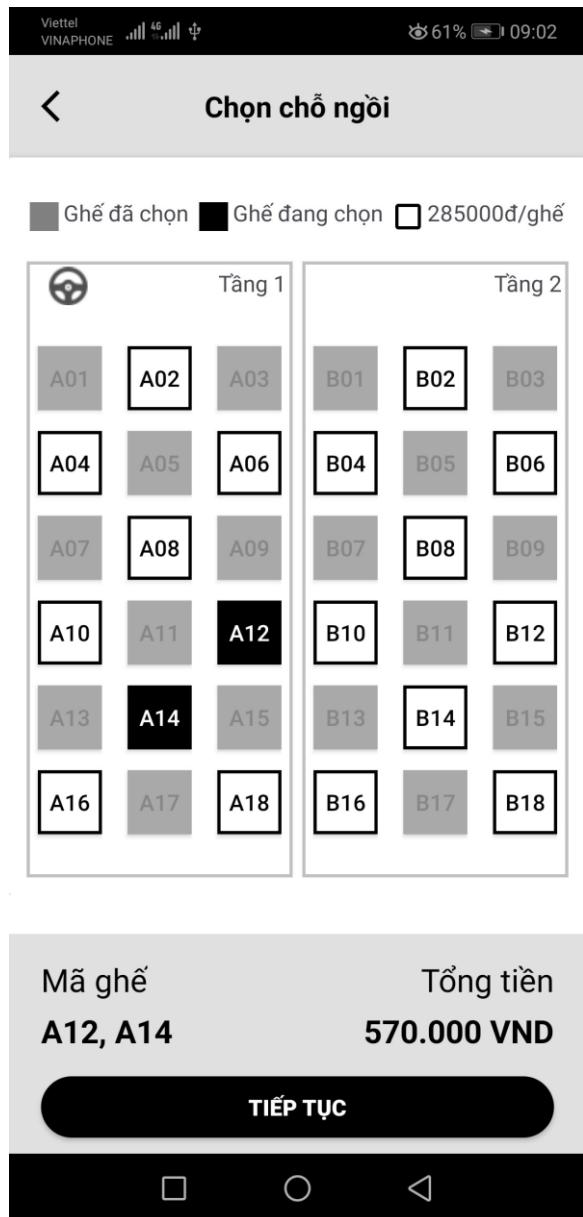


```
59 @ ...
60     public ChooseCarrierViewModel(Context context, ActivityChooseCarrierBinding activityChooseCarrierBinding) {
61         mListRoutes = new ArrayList<Routes>();
62         adapter = new ChooseCarrierAdapter(context, mListRoutes); // khởi tạo đối tượng ChooseCarrierAdapter
63         activityChooseCarrierBinding.rcvCarrier.setAdapter(adapter); // đặt adapter cho rcv trong activityChooseCarrierBinding
64         String origin = SharedPrefOriginDestination.getOrigin(context);
65         String destination = SharedPrefOriginDestination.getDestination(context);
66         String TVDate = SharedPrefOriginDestination.getDate(context);
67         String token = SharedPreferencesUtil.getToken(context);
68         iLoginService = ApiUtils.getApiService(token);
69         iLoginService.getRoutesProvinces(origin, destination, TVDate)
70             .enqueue(new Callback<RoutesResponse<Routes>>() {
71                 @Override
72                 public void onResponse(Call<RoutesResponse<Routes>> call, Response<RoutesResponse<Routes>> response) {
73                     if (response.isSuccessful()) {
74                         routesResponse = response.body();
75                         List<Routes> mListRoutes3 = new ArrayList<>();
76                         mListRoutes3 = routesResponse.getData();
77                         mListRoutes = (ArrayList<Routes>) routesResponse.getData();
78                         adapter.notifyDataSetChanged();
79                         if (mListRoutes.size() > 0) {
80                             activityChooseCarrierBinding.rcvCarrier.setVisibility(View.VISIBLE);
81                             activityChooseCarrierBinding.noCarrier.setVisibility(View.GONE);
82                         } else {
83                             activityChooseCarrierBinding.rcvCarrier.setVisibility(View.GONE);
84                         }
85                     }
86                 }
87             });
88     }
89 }
```

Hình 103. Hàm thực hiện call api để lấy dữ liệu nhà xe theo ngày đi, điểm đi, điểm đến.

6.1.6. Màn hình chọn chỗ ngồi.

Code giao diện bằng xml và xử lý chức năng hiển thị dữ liệu các button, hiển thị mã ghế đã chọn và tính tổng tiền.



Hình 104. Màn hình chọn chỗ ngồi.

Nhận dữ liệu từ ChooseCarrierActivity thực hiện tách dữ liệu ra 2 ArrayList để đổ vào adapter.

```

Intent intent = getIntent();
Routes routes = Parcels.unwrap(intent.getParcelableExtra("routes"));
int count = 0;
ArrayList<Seat> allSeat1 = new ArrayList<>();
ArrayList<Seat> allSeat2 = new ArrayList<>();
ArrayList<Seat> filteredList1 = new ArrayList<>();
ArrayList<Seat> filteredList2 = new ArrayList<>();
if (!routes.getTrips().getSeats().isEmpty()) {
    for (Seat seat : routes.getTrips().getSeats()) {
        if (count < 18) {
            filteredList1.add(seat);
            count++;
        } else {
            filteredList2.add(seat);
        }
    }
    allSeat1.clear();
    allSeat2.clear();
    allSeat1.addAll(filteredList1);
    allSeat2.addAll(filteredList2);
}
adapter1 = new ChooseSeatAdapter(allSeat1, context: this, chooseSeatViewModel);
adapter2 = new ChooseSeatAdapter(allSeat2, context: this, chooseSeatViewModel);
activityChooseSeatBinding.recyclerViewGroup1.setLayoutManager(new GridLayoutManager(context: this, spanCount: 3));
activityChooseSeatBinding.recyclerViewGroup2.setLayoutManager(new GridLayoutManager(context: this, spanCount: 3));
activityChooseSeatBinding.recyclerViewGroup1.setAdapter(adapter1);
activityChooseSeatBinding.recyclerViewGroup2.setAdapter(adapter2);

```

Hình 105. Xử lý dữ liệu để đổ dữ liệu vào adapter hiển thị trạng thái ghế lên giao diện.

Xử lý chọn và bỏ chọn ghế

```

public void onSeatSelected(Seat seat) {
    if (seat.getStatusSeat()) {
        // Ghế đã bị chọn
        return;
    }

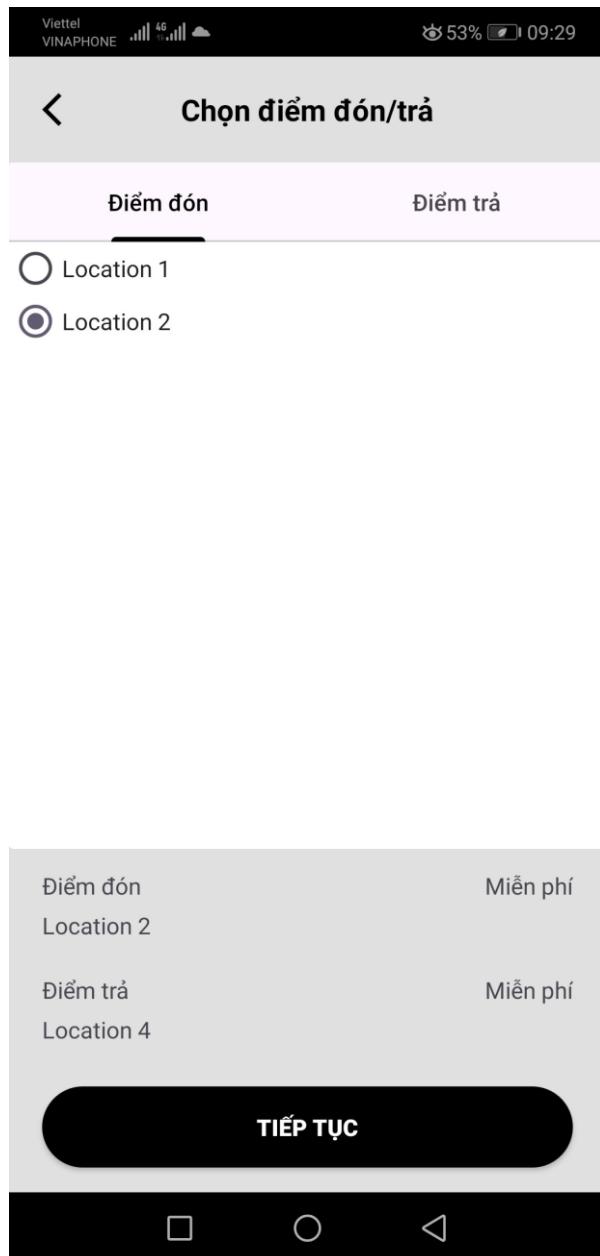
    if (selectedSeats.contains(seat)) {
        // Bỏ chọn ghế nếu đã chọn
        selectedSeats.remove(seat);
        selectedSeatCount.setValue(selectedSeatCount.getValue() - 1);
        adapter1.updateSeatStatus(seat);
        adapter2.updateSeatStatus(seat);
    } else {
        // Chọn ghế nếu chưa chọn
        selectedSeats.add(seat);
        selectedSeatCount.setValue(selectedSeatCount.getValue() + 1);
    }
    // Cập nhật danh sách ghế đã chọn và tính tổng giá tiền
    updateSelectedSeatsText();
    updateTotalPrice();
}

```

Hình 106. Xử lý chọn và bỏ chọn ghế.

6.1.7. Màn hình chọn điểm đón, điểm trả.

Code giao diện bằng xml và xử lý chức năng hiển thị các vị trí bằng radio button. Xử lý khi người dùng chọn sẽ hiển thị vị trí đã chọn trên màn hình.



Hình 107. Màn hình chọn điểm đón, điểm trả.

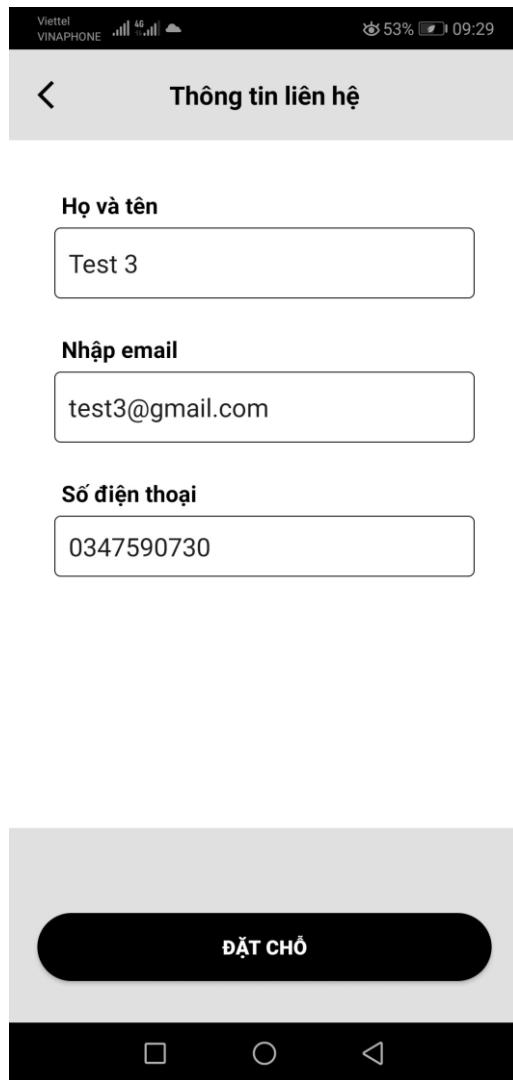
Nhận dữ liệu từ ChooseSeatActivity và xử hiển thị RadioButton dữ liệu lên 2 Tab view điểm đón, điểm trả.

```
ArrayList<String> listLocation1 = getIntent().getStringArrayListExtra("listLocation1");
ArrayList<String> listLocation2 = getIntent().getStringArrayListExtra("listLocation2");
// Cập nhật danh sách cho ViewModel
viewModel.setLocations1(listLocation1);
viewModel.setLocations2(listLocation2);
// Tạo một TabLayout và một ViewPager
TabLayout tabLayout = binding.tabLayout;
ViewPager viewPager = binding.viewPager;
// Tạo một TabViewAdapter để quản lý các tab
TabViewAdapter adapter = new TabViewAdapter(getSupportFragmentManager());
// Thêm 2 tab với tên Điểm đón và Điểm trả
adapter.addFragment(new TabViewFragment(), "Điểm đón");
adapter.addFragment(new TabViewFragment2(), "Điểm trả");
// Gán adapter cho viewPager
viewPager.setAdapter(adapter);
viewPager.setAdapter(adapter);
// Đóng bộ tabLayout với viewPager
tabLayout.setupWithViewPager(viewPager);
```

Hình 108. Xử lý hiển thị điểm đón điểm trả trên 2 Tab View.

6.1.8. Màn hình nhập thông tin liên hệ.

Code giao diện bằng xml và xử lý hiển thị dữ liệu thông tin người dùng.



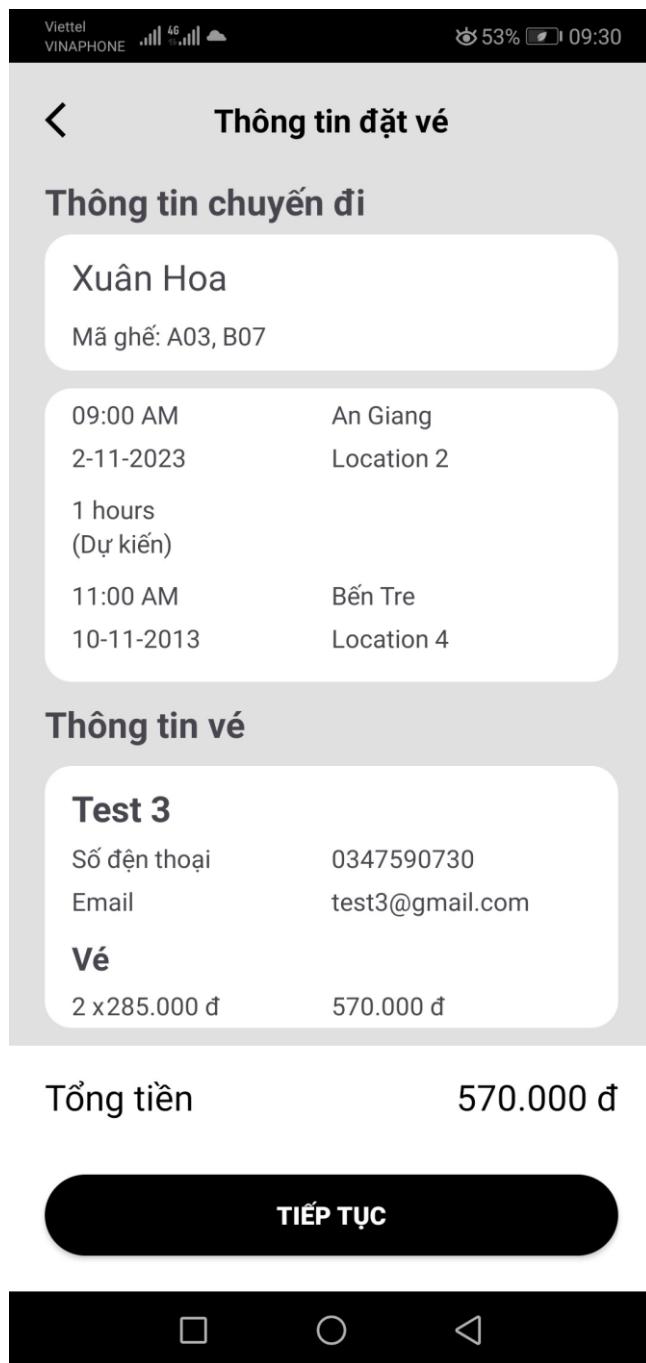
Hình 109. Màn hình nhập thông tin liên hệ.

Lấy dữ liệu được lưu Share Preferences hiển thị lên giao diện.

```
String name = SharedPrefUser.getName( context: this);
String email = SharedPrefUser.getEmail( context: this);
String phone = SharedPrefUser.getPhone( context: this);
viewModel.setName(name);
viewModel.setEmail(email);
viewModel.setPhone(phone);
```

6.1.9. Màn hình hiển thị dữ liệu thông tin đặt vé.

Code giao diện bằng xml và hiện thị dữ liệu thông tin đặt vé.



Hình 110. Màn hình hiển thị dữ liệu thông tin đặt vé.

Lấy dữ liệu khi người dùng chọn và hiển thị lên giao diện thông tin đặt vé. Sau khi người dùng nhấn nút tiếp tục thì gọi hàm updateSeat và gọi hàm insertBookingSeat.

```

public void updateSeat() {
    Trip trips = new Trip();
    String originTime = routesLiveData.getValue().getTrips().getOriginTime();
    String destinationTime = routesLiveData.getValue().getTrips().getDestinationTime();
    String originDate = routesLiveData.getValue().getTrips().getOriginDate();
    String destinationDate = routesLiveData.getValue().getTrips().getDestinationDate();
    List<Seat> seats = routesLiveData.getValue().getTrips().getSeats();
    int availableSeats = routesLiveData.getValue().getTrips().getAvailableSeats() - getSize(
        trips.setOriginTime(originTime);
        trips.setDestinationTime(destinationTime);
        trips.setOriginDate(originDate);
        trips.setDestinationDate(destinationDate);
        trips.setSeats(seats);
        trips.setAvailableSeats(availableSeats);
        routes.setTrips(trips);
        routes.setId(routesLiveData.getValue().get_id());
        routes.setPrice(routesLiveData.getValue().getPrice());
        String token1 = SharedPreferencesUtil.getToken(context);
        iLoginService = ApiUtils.getApiService(token1);
        ↳ LAPTOP-38ED4DO3\ASUS *

        String token1 = SharedPreferencesUtil.getToken(context);
        iLoginService = ApiUtils.getApiService(token1);
        ↳ LAPTOP-38ED4DO3\ASUS *
        iLoginService.updateSeat(routes).enqueue(new Callback<Routes>() {
            ↳ LAPTOP-38ED4DO3\ASUS *
            @Override
            public void onResponse(Call<Routes> call, Response<Routes> response) {
                if (response.isSuccessful()) {
                } else {
                    errorMessage.setValue("Error update booking seat: " + response.errorBody().toString());
                    Log.d(tag: "ERRORUpdate", response.errorBody().toString());
                }
            }
            ↳ LAPTOP-38ED4DO3\ASUS
            @Override
            public void onFailure(Call<Routes> call, Throwable t) {
                Toast.makeText(getContext(), text: "update thất bại", Toast.LENGTH_SHORT).show();
            }
        });
    }
}

```

Hình 111. Hàm updateSeat.

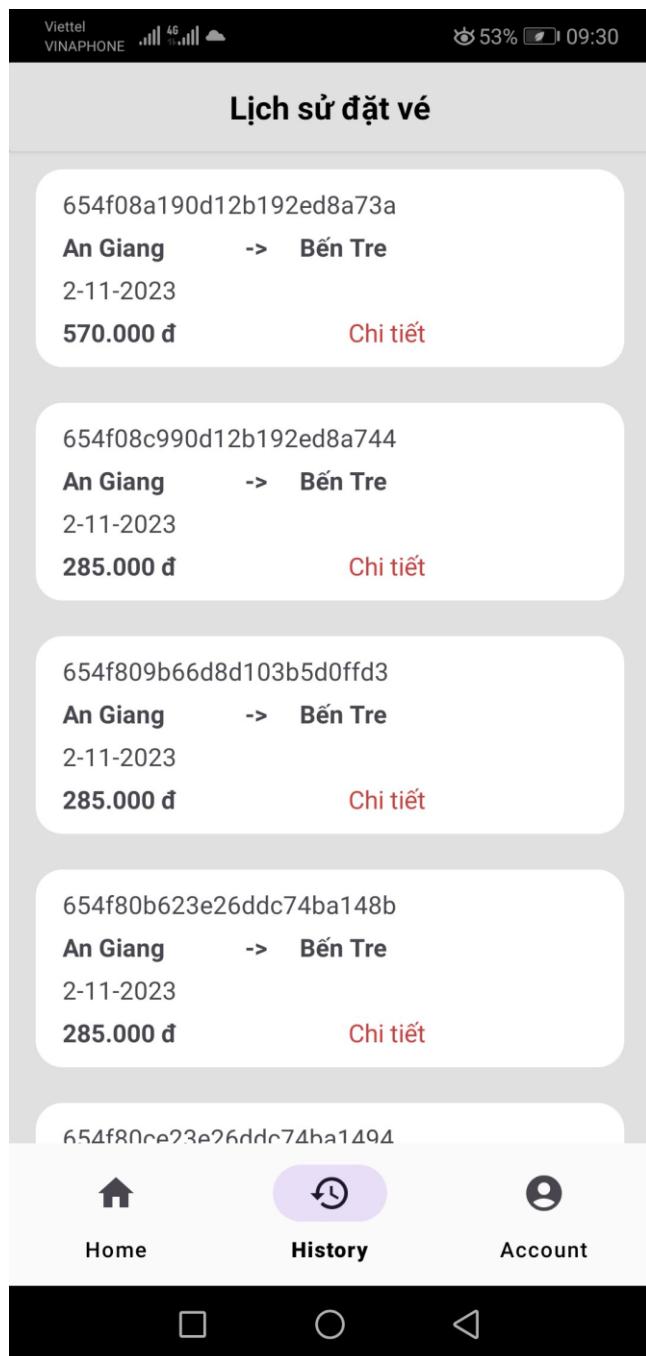
Hàm InsertBookingSeat.

```
String token1 = SharedPreferencesUtil.getToken(context);
iLoginService = ApiUtils.getApiService(token1);
@ LAPTOP-38ED4DO3\ASUS *
iLoginService.updateSeat(routes).enqueue(new Callback<Routes>() {
    @ LAPTOP-38ED4DO3\ASUS *
    @Override
    public void onResponse(Call<Routes> call, Response<Routes> response) {
        if (response.isSuccessful()) {
        } else {
            errorMessage.setValue("Error update booking seat: " + response.errorBody().toString());
            Log.d(tag: "ERRORUpdate", response.errorBody().toString());
        }
    }
    @ LAPTOP-38ED4DO3\ASUS
    @Override
    public void onFailure(Call<Routes> call, Throwable t) {
        Toast.makeText(getContext(), text: "update thất bại", Toast.LENGTH_SHORT).show();
    }
});
```

Hình 112. Hàm InsertBookingSeat.

6.1.10. Màn hình lịch sử đặt vé.

Code giao diện bằng xml và thực hiện chức năng get data từ api để hiển thị danh sách lịch sử đặt vé.



Hình 113. Màn hình lịch sử đặt vé.

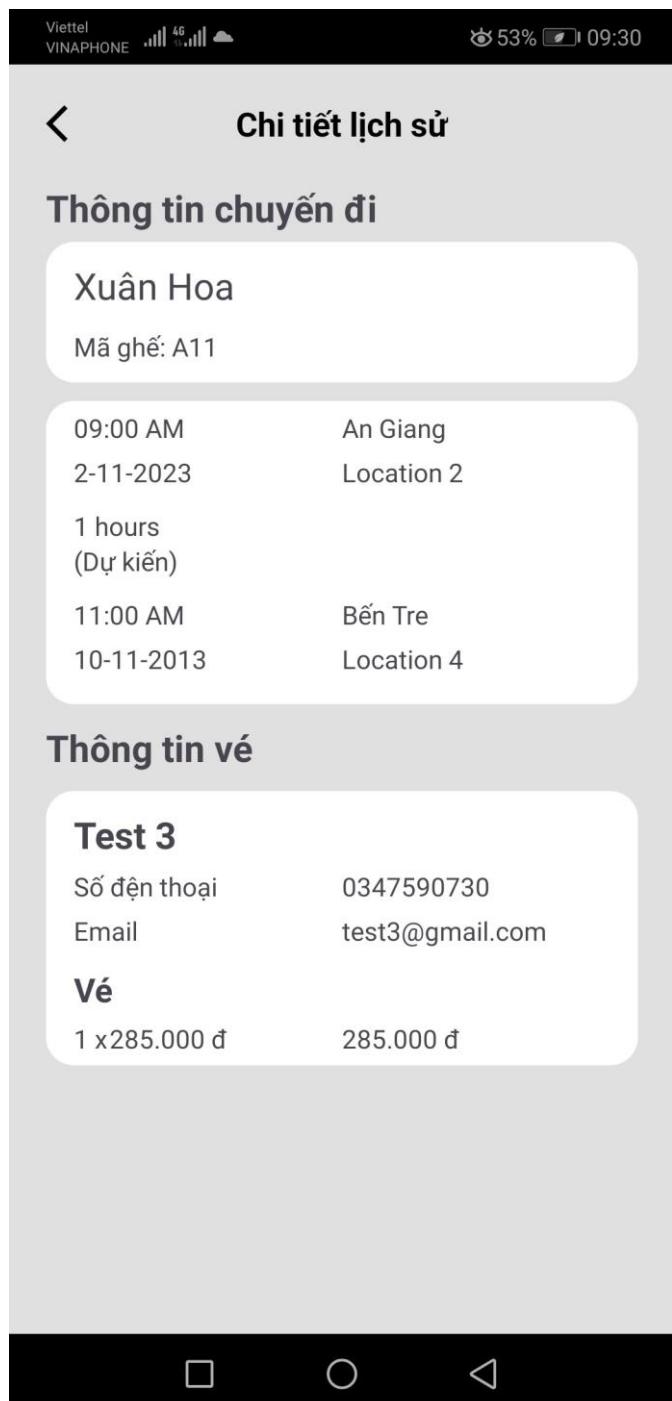
Hàm lấy dữ liệu từ api để hiển thị danh sách lịch sử đặt vé.

```
public HistoryViewModel(Context context, FragmentHistoryBinding binding) {
    mListBookingSeats = new ArrayList<>();
    adapter = new HistoryAdapter(context, mListBookingSeats);
    binding.rcvCarrierHistory.setAdapter(adapter);
    String token = SharedPreferencesUtil.getToken(context);
    String id = "65294bbf2fb2ab3bb6423181";
    String customerId = SharedPrefUser.getId(context);
    iLoginService = ApiUtils.getApiService(token);
    iLoginService.getBookingSeat(customerId).enqueue(new Callback<BookingSeatResponse<BookingSeat>>() {
        @LAPTOP-38ED4D03\ASUS
        @Override
        public void onResponse(Call<BookingSeatResponse<BookingSeat>> call, Response<BookingSeatResponse<BookingSeat>> response) {
            if (response.isSuccessful()) {
                bookingSeatResponse = response.body();
                mListBookingSeats = (ArrayList<BookingSeat>) bookingSeatResponse.getData();
                adapter.notifyDataSetChanged();
            } else {
                Log.d("BookingSeatResponse12", "Lỗi khi lấy dữ liệu từ API");
            }
        }
        @LAPTOP-38ED4D03\ASUS
        @Override
        public void onFailure(Call<BookingSeatResponse<BookingSeat>> call, Throwable t) {
            try {
                Log.d("BookingSeatResponse13", "Lỗi kết nối API");
                Log.d("ErrorConnectAPIBookingSeatResponse", t.getMessage());
            } catch (Exception e) {
                Log.d("ExceptionOnFailureBookingSeatResponse", e.getMessage());
                System.out.println("Message ExceptionOnFailure: " + e.getMessage());
            }
        }
    });
}
```

Hình 114. Hàm lấy dữ liệu từ api lịch sử đặt vé.

6.1.11. Màn hình hiển thị thông tin chi tiết lịch sử đặt vé.

Code giao diện bằng xml và hiển thị thông tin chi tiết vé xe.



Hình 115. màn hình chi tiết lịch sử đặt vé.

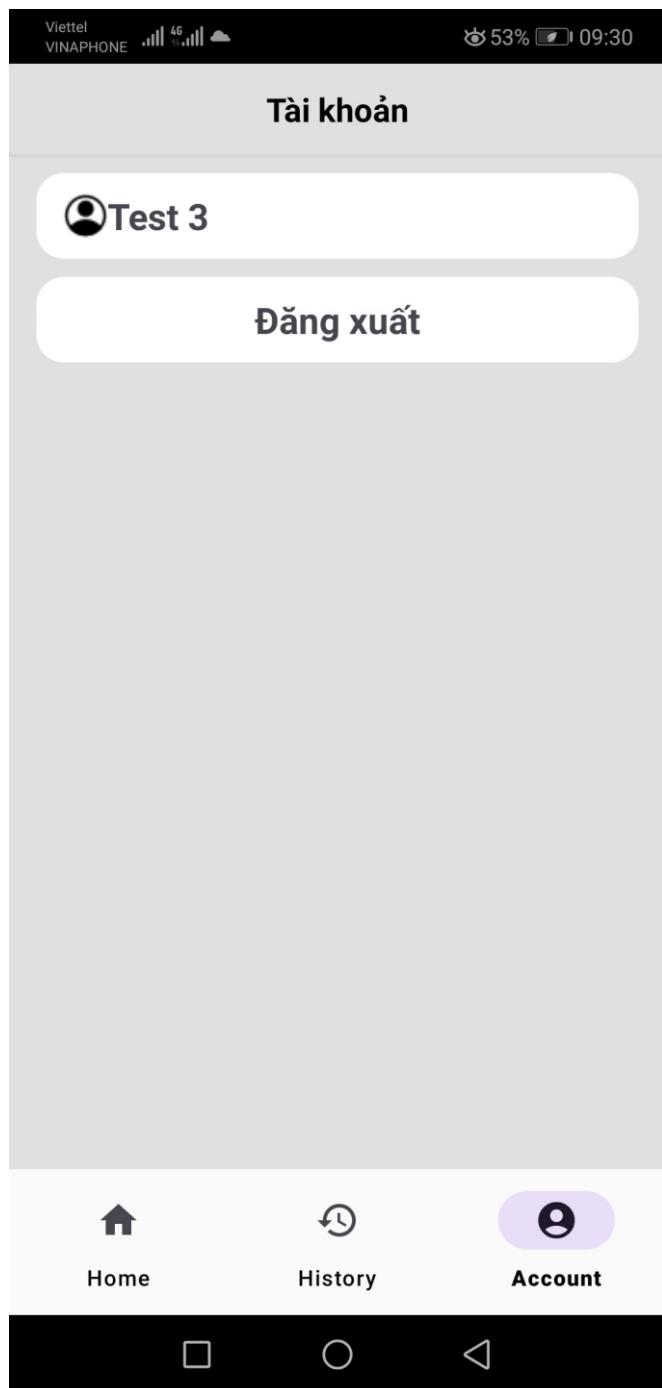
Xử lý hiển thị chi tiết thông tin lịch sử đặt vé.

```
public class DetailHistoryViewModel extends ViewModel {  
    2 usages  
    private BookingSeat bookingSeat;  
  
    2 usages  
    private MutableLiveData<String> seat = new MutableLiveData<>();  
  
    17 usages  ↳ LAPTOP-38ED4DO3\ASUS  
    public BookingSeat getBookingSeat() { return bookingSeat; }  
  
    17 usages  ↳ LAPTOP-38ED4DO3\ASUS  
    public void setBookingSeat(BookingSeat bookingSeat) { this.bookingSeat = bookingSeat; }  
    ↳  
    ↳ LAPTOP-38ED4DO3\ASUS  
    public MutableLiveData<String> getSeat() { return seat; }  
  
    1 usage  ↳ LAPTOP-38ED4DO3\ASUS  
    public void updateSeatIds(ArrayList<BookingSeatDetails> selectedSeats) {  
        StringBuilder builder = new StringBuilder();  
        for (BookingSeatDetails seat : selectedSeats) {  
            builder.append(seat.getSeatNumber()).append(", ");  
        }  
        if (builder.length() > 0) {  
            builder.setLength(builder.length() - 2); // Xóa dấu phẩy cuối cùng  
        }  
        seat.setValue(builder.toString());  
    }  
}
```

Hình 116. Xử lý hiển thị chi tiết thông tin lịch sử đặt vé.

6.1.12. Màn hình tài khoản.

Code giao diện bằng xml và xử lý chức năng đăng xuất.



Hình 117. Màn hình tài khoản.

Xử lý hiển thị tên người dùng và đăng xuất

```
7 usages  ↳ LAPTOP-38ED4DO3\ASUS
public class AccountViewModel extends ViewModel {
    2 usages
    private String name = new MutableLiveData<>().toString();

    ↳ LAPTOP-38ED4DO3\ASUS
    public String getName() {
        return name;
    }

    ↳ LAPTOP-38ED4DO3\ASUS
    public void setName(String name) { this.name = name; }
}

String name = SharedPrefUser.getName(getContext());
viewModel.setName(name);
binding.setPresenter(logout() → {
    Intent intent = new Intent(getActivity(), LoginActivity.class);
    getActivity().startActivity(intent);
    ((Activity) getActivity()).finish();
});
```

Hình 118. Xử lý hiển thị tên người dùng và đăng xuất.

6.2. Kiểm thử

STT	Chức năng	Số lần chạy	Số lần thành công	Số lần không thành công	Nhận xét
1	Đăng ký	10	10	0	Tốt
2	Đăng nhập	10	10	0	Tốt
3	Đăng Xuất	10	10	0	Tốt
4	Đặt vé xe	10	10	0	Tốt
5	Xem lịch xử đặt vé	10	10	0	Tốt

Bảng 31. Kiểm thử

Chương 7: Kết luận và hướng mở rộng.

7.1. Kết quả thu được.

Đồ án cơ bản đã hoàn thiện với kết quả là xây dựng hỗ trợ cho người dùng có thể đặt vé xe online một cách tiện lợi và nhanh chóng. Các chức năng thu được như sau: Đăng nhập, đăng ký, đăng xuất, đặt vé xe và xem lịch sử đặt vé.

7.2. Hướng phát triển trong tương lai.

Phát triển thêm chức năng thanh toán qua ví điện tử momo.

7.3. Kết luận.

Trong quá trình xây dựng ứng dụng đặt vé xe khách, em đã đạt được nhiều thành tựu quan trọng và học được nhiều kiến thức mới. Em đã thành công trong việc phát triển một giao diện người dùng thân thiện và dễ sử dụng, tối ưu hóa trải nghiệm người dùng.

Tuy nhiên, quá trình phát triển cũng đặt ra một số thách thức quan trọng. Việc quản lý dữ liệu và bảo mật thông tin cá nhân là một vấn đề cần chú ý, và Em đã học được rất nhiều về cách xử lý vấn đề này một cách hiệu quả.

Quá trình xây dựng ứng dụng đặt vé xe khách là một hành trình đầy thách thức, nhưng cũng mang lại nhiều kinh nghiệm quý báu. Em tin rằng ứng dụng sẽ đóng góp tích cực vào việc cung cấp dịch vụ vận chuyển hiệu quả và thuận tiện cho người sử dụng.

7.4. Link source code:

Link: <https://github.com/ydang2002/CoachTicket>

Tài liệu tham khảo.

- [1] Giang Phan, “Java là gì? Tổng quan về ngôn ngữ lập trình java”, TOPDev, <https://topdev.vn/blog/tong-quan-ve-ngon-nu-lap-trinh-java/#java-la-gi>, Truy cập ngày 15 tháng 9 năm 2023.
- [2] Chu Thị Thơm, “Bắt đầu với NoSQL và MongoDB”, VIBLO, <https://viblo.asia/p/bat-dau-voi-nosql-va-mongodb-jvEla00zZkw>, Truy cập ngày 15 tháng 9 năm 2023.
- [3] Đông Tùng, “NodeJS là gì? Đặc điểm và ứng dụng của Node.JS”, Wiki Tino, <https://wiki.tino.org/nodejs-la-gi/#nodejs-la-gi>, Truy cập ngày 15 tháng 9 năm 2023.
- [4] Lê Đức Mạnh, “RESTful API là gì?”, VIBLO, <https://viblo.asia/p/restful-api-la-gi-1Je5EDJ4lnL>, Truy cập ngày 15 tháng 9 năm 2023.
- [5] Kiều Việt Anh, “Retrofit là gì và cách sử dụng - Caching dữ liệu với Retrofit - Lưu dữ liệu offline”, VIBLO, <https://viblo.asia/p/retrofit-la-gi-va-cach-su-dung-caching-du-lieu-voi-retrofit-luu-du-lieu-offline-XL6lAN2R5ek>, Truy cập ngày 15 tháng 9 năm 2023.
- [6] Lê Công Hậu, “Kiến trúc dự án trong Android – MVVM với Data Binding”, WordPress, <https://atozit427053812.wordpress.com/2018/10/17/phan-3-kien-truc-du-an-trong-android-mvvm-voi-data-binding/>, Truy cập ngày 15 tháng 9 năm 2023.
- [7] Hayk Simonyan, “Deploying a NestJS app for Free on Cyclic”, LinkedIn, <https://www.linkedin.com/pulse/deploying-nestjs-app-free-oncyclic-hayk-simonyan#:~:text=Cyclic%20is%20a%20cloud%20platform,runn,ing%20all%20the%20time.>, Truy cập ngày 15 tháng 9 năm 2023.