TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA ĐIỆN – ĐIỆN TỬ

LÊ VÕ HOÀNG THÔNG

BÃI ĐẬU XE THÔNG MINH

ĐỒ ÁN TỔNG HỢP
KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA ĐIỆN – ĐIỆN TỬ

LÊ VÕ HOÀNG THÔNG
41900278

BÃI ĐẬU XE THÔNG MINH

ĐỒ ÁN TỔNG HỢP
KỸ THUẬT ĐIỆN TỬ - VIỄN THÔNG

Người hướng dẫn
TS. Trần Thành Nam

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Tôi xin chân thành cảm ơn đến trường Đại học Tôn Đức Thắng và Khoa Điện – Điện tử đã tạo điều kiện tốt nhất về cơ sở vật chất để tôi có thể tìm kiếm và nghiên cứu tài liệu phục vụ cho việc thực hiện đồ án.

Tôi xin gửi lời cảm ơn chân thành và sâu sắc nhất đến với thầy TS. Trần Thành Nam đã nhiệt tình hướng dẫn, hỗ trợ, giúp tôi có một nền tảng kiến thức vững chắc để có thể thực hiện đồ án này một cách chỉnh chu và tốt nhất.

Trong quá trình thực hiện sẽ không thể tránh khỏi những sai sót về mặt kiến thức cũng như về cách trình bày báo cáo. Chính vì vậy, tôi rất mong nhận được sự nhận xét, góp ý để giúp tôi có thể hoàn thiện bài báo cáo này.

Tôi xin chân thành cảm ơn.

TP. Hồ Chí Minh, ngày tháng năm 2024

Tác giả

Lê Võ Hoàng Thông

Công trình được hoàn thành tại Trường Đại học Tôn Đức Thắng

Cán bộ hướng dẫn khoa học: .....

.....

(Ghi rõ học hàm, học vị, họ tên và chữ ký)

Đồ án tốt nghiệp/tổng hợp được bảo vệ tại **Hội đồng đánh giá Đồ án tốt nghiệp/tổng hợp của Trường Đại học Tôn Đức Thắng** vào ngày… /…/……

Xác nhận của Chủ tịch Hội đồng đánh giá Đồ án tốt nghiệp/tổng hợp và Trưởng khoa quản lý chuyên ngành sau khi nhận Đồ án tốt nghiệp/tổng hợp đã được sửa chữa (nếu có).

CHỦ TỊCH HỘI ĐỒNG	**TRƯỞNG KHOA**
……	……

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của thầy TS. Trần Thành Nam. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong Đồ án tốt nghiệp/ tổng hợp còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung Đồ án tốt nghiệp/ tổng hợp của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm 2024

Tác giả

(ký tên và ghi rõ họ tên)

Lê Võ Hoàng Thông

BÃI ĐẬU XE THÔNG MINH
TÓM TẮT

Sự phát triển không ngừng của nền kinh tế thế giới kéo theo tốc độ phát triển của khoa học – kỹ thuật đã khiến cuộc sống của con người ngày càng trở nên tiện nghi, hiện đại hơn. Ngày nay, mật độ dân cư tại các thành phố lớn ngày càng trở nên đông đúc, đi cùng với đó sự gia tăng về xe cộ, đặc biệt là xe ô tô. Tình trạng xe ô tô xuất hiện ngày một nhiều tại các thành phố lớn góp phần phản ánh về sự phát triển của một quốc gia. Chính vì điều này đã phát sinh ra những vấn đề về ô nhiễm môi trường, ùn tắc giao thông, thiếu bãi đậu xe đang cần được cấp thiết giải quyết.

Công nghệ điện tử và tự động hoá đã khiến nhiều vấn đề được giải quyết một cách nhanh chóng. Các bãi đậu xe truyền thống đã không còn phù hợp để sử dụng tại các toà nhà, chung cư, trường học,… vì những rắc rối mà nó mang lại như: tốn chi phí nhân công trông giữ, tình trạng chờ đợi lấy thẻ xe và thu phí khi ra vào bãi xe quá lâu. Đề tài "Bãi đậu xe thông minh" được lựa chọn để nghiên cứu với các tiện ích mà nó mang lại như đăng ký dễ dàng, quét mã QR nhanh chóng ngay trên ứng dụng thiết kế để ra vào cổng, biết được tình trạng và vị trí bãi đậu xe nhằm để tiết kiệm thời gian như một lời giải cấp thiết cho bài toán này.

SMART PARKING PLACE

ABSTRACT

The continuous development of the world economy and the rapid development of science and technology have made people's lives more comfortable and modern. Nowadays, population density in big cities is becoming increasingly crowded, along with an increase in vehicles, especially cars. The increasing presence of cars in big cities reflects the development of a country. Because of this, problems of environmental pollution, traffic congestion, and lack of parking have arisen that need to be urgently resolved.

With the increasingly developing level of science and technology, many problems have been solved quickly, thanks to electronic technology and automation. Traditional parking lots are no longer suitable for use in buildings, apartments, schools, etc. because of the problems they bring such as high labor costs for parking, long waiting time to get parking cards and fees when entering and exiting the parking lot. The project "Smart Parking Place" was chosen for research due to its benefits, such as easy registration, quick QR code scanning directly within the app for entry and exit, and real-time updates on parking lot status and location, which provide a timely solution to this issue.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC CÁC CHỮ VIẾT TẮT

ACK	Acknowledged
NACK	Not Acknowledged
QR	Quick Response
SPI	Serial Peripheral Interface
UART	Universal Asynchronous Receiver - Transmitter
USB	Universal Serial Bus

CHƯƠNG 1. GIỚI THIỆU ĐỀ TÀI

1.1 Mục đích đề tài

Dựa trên số liệu thống kê mới nhất từ Hiệp hội các nhà sản xuất ô tô Việt Nam (VAMA) trong tháng 05/2024 doanh số bán hàng của toàn thị trường đạt 25794 xe, tăng 24% so với tháng 05/2023. VinFast – một nhà sản xuất ô tô điện cũng ghi nhận doanh thu tăng trưởng 269.7% trong quý I năm 2024 so với cùng kỳ năm 2023. Chính những con số trên đã cho chúng ta thấy rằng lượng xe ô tô bán ra đang ngày một tăng dần trong mỗi năm do mức sống của con người ngày càng được nâng cao, bên cạnh đó, vấn đề xây dựng các bãi đậu xe đang được đặt ra để thuận tiện cho người dân trong công việc cũng như đi lại của họ.

Ngoài việc xây dựng các bãi đậu xe ra thì các yêu cầu về việc áp dụng khoa học – kỹ thuật sẽ giúp cho bãi đậu xe trở nên được tự động, việc quản lý số lượng xe trong bãi sẽ trở nên thông minh, giúp tối ưu hoá thời gian cũng như là chi phí cho người dân. Có thể thấy, bãi đậu xe truyền thống đang còn tồn đọng nhiều bất cập chẳng hạn như tốn quá nhiều nhân công trong việc trông giữ xe, việc ghi phiếu giữ xe bằng giấy dẫn đến người dân dễ làm mất phiếu hoặc ướt do trời mưa khiến nội dung trên phiếu trở nên không rõ ràng, tiềm ẩn nhiều rủi ro trong việc quản lý xe ra vào bãi.

Hình 1-1: Phiếu giữ xe tại bãi giữ xe truyền thống

Theo Tổng cục Thống kê, dân số đô thị tại Việt Nam đã tăng nhanh trong những năm gần đây, đặc biệt là tại Hà Nội và TP. Hồ Chí Minh. Hà Nội và TP. Hồ Chí Minh đều có hàng triệu xe máy và hàng trăm nghìn ô tô đang lưu thông hàng ngày đã dẫn đến mật độ giao thông tăng cao, tình trạng ùn tắc thường xuyên, đặc biệt trong giờ cao điểm, các tuyến đường chính thường xuyên bị kẹt xe nghiêm trọng, gây mất thời gian và ảnh hưởng đến chất lượng cuộc sống. Mặc dù có nhiều dự án phát triển cơ sở hạ tầng như mở rộng đường sá, xây dựng cầu vượt và phát triển hệ thống giao thông công cộng nhưng tốc độ tăng trưởng phương tiện giao thông vẫn vượt xa sự phát triển của hạ tầng. Nhiều tuyến đường và nút giao thông tại các thành phố lớn chưa đáp ứng được nhu cầu thực tế, gây ra ùn tắc và tai nạn giao thông.

Hình 1-2: Tình trạng đỗ ô tô vào mỗi dịp lễ

Các bãi đậu xe truyền thống tại các thành phố lớn ở Việt Nam đang gặp phải nhiều khó khăn và hạn chế chẳng hạn như diện tích đất dành cho bãi đậu xe ở các khu đô thị lớn rất hạn chế do giá đất cao và ưu tiên sử dụng đất cho các mục đích khác như xây dựng nhà ở, văn phòng và trung tâm thương mại. Nhiều khu vực trung tâm thành phố gần như không có bãi đậu xe công cộng hoặc có nhưng lại không thể đáp ứng được nhu cầu, gây khó khăn cho người dân và doanh nghiệp. Các bãi đậu xe truyền thống đòi hỏi nhiều nhân công để giám sát, thu phí và điều hành hoạt động hàng ngày, dẫn đến chi phí vận hành cao, bên cạnh đó, việc thu phí và quản lý thủ công dễ dẫn đến sai sót và thất thoát doanh thu. Người đậu xe phải chờ đợi rất lâu để lấy thẻ xe và

thanh toán khi ra vào bãi đậu, gây mất thời gian và phiền toái. Trong giờ cao điểm, tình trạng xếp hàng dài để vào và ra khỏi bãi đậu xe là rất phổ biến. Hệ thống giám sát và bảo vệ ở các bãi đậu xe truyền thống thường chưa đủ hiện đại và hiệu quả, dễ dẫn đến tình trạng mất cắp xe và tài sản cá nhân. Bãi đậu xe truyền thống thường không được quy hoạch hợp lý, gây ảnh hưởng đến môi trường đô thị như tạo ra các điểm nóng ô nhiễm không khí và tiếng ồn. Việc không có hệ thống quản lý thông minh còn làm tăng lượng khí thải do xe phải di chuyển lâu hơn để tìm chỗ đậu.

Đề tài "Bãi đậu xe thông minh" được ra đời nhằm khắc phục những khuyết điểm trên thông qua việc ứng dụng công nghệ IoT. Chỉ với việc đăng ký ngay trên ứng dụng, người dùng giờ đây có thể xem được vị trí các bãi đậu xe xung quanh, tình trạng bãi đậu, đặt chỗ và tính phí ngay trên ứng dụng một cách dễ dàng, ra vào bãi thông qua việc quét mã QR ngay trên ứng dụng thiết kế giúp việc quản lý thông tin khách hàng trở nên dễ dàng và bảo mật. Với những tiện ích mà bãi đậu xe thông minh này mang lại, mức độ an toàn cũng sẽ được nâng cao hơn, giúp tiết kiệm thời gian và hạn chế được các hành vi giả mạo.

1.2 Yêu cầu của đề tài

Bãi đậu xe thông minh sẽ có tối đa là 4 chỗ đậu xe với 4 cảm biến được đặt tại 4 vị trí đậu để có thể cập nhật tình trạng xe đậu và gửi dữ liệu về web server. Một cảm biến được bố trí ở lối vào và một cảm biến khác được bố trí lối ra đều có thanh chắn ở phía trước nhằm xác định có xe ra vào để thực hiện yêu cầu quét mã QR, nếu đúng là mã QR được tạo từ ứng dụng thì hệ thống sẽ thực hiện mở thanh chắn cho xe ra vào bãi. Một màn hình ở phía trước được sử dụng để hiển thị chi tiết về tình trạng của 4 chỗ đậu xe và đưa ra yêu cầu thực hiện quét mã QR khi phát hiện có người dùng vào và ra bãi đậu xe. Máy quét mã QR sẽ được thiết kế cạnh màn hình để người dùng có thể thực hiện quét mã QR tạo từ ứng dụng khi có yêu cầu quét được đưa ra.

Ứng dụng được thiết kế cho bãi đậu xe thông minh sẽ có một giao diện để người dùng có thể đăng ký tài khoản và đăng nhập vào hệ thống. Sau khi đã đăng nhập vào hệ thống thì người dùng sẽ vào giao diện chính của ứng dụng với các tính năng tìm bãi

xe, nạp tiền, thanh toán và đăng xuất khỏi tài khoản. Tính năng nạp tiền vào hệ thống để người dùng có thể thực hiện các bước đặt chỗ và thanh toán phí đậu xe, khi không nạp tiền vào hệ thống thì người dùng sẽ không thể thực hiện được thao tác đặt chỗ trong bãi đậu xe. Người dùng có thể tìm được vị trí bãi đậu xe gần với mình nhất thông qua giao diện tìm bãi đậu xe với các Marker đại diện cho các địa điểm bãi đậu xe được gắn trên bản đồ được cung cấp bởi Google. Khi đã xác định được vị trí bãi đậu xe mà người dùng muốn đậu, ứng dụng sẽ hiển thị tình trạng của 4 chỗ tại vị trí mà người dùng đã chọn trước đó, giúp người dùng có thể theo dõi được chính xác tình trạng chỗ theo thời gian thực để đưa ra lựa chọn đặt chỗ hoặc tìm một bãi đậu xe khác. Một giao diện đặt chỗ sẽ cho phép người dùng nhập vào các thông tin như vị trí đậu, ngày đậu và thời gian bắt đầu đậu nhằm có thể giúp ứng dụng xác định và ưu tiên đối với người đặt chỗ trước, tránh cho người dùng khác đặt cùng một chỗ với thời gian trùng lặp nhau.

Dựa vào các thông tin đặt chỗ mà người dùng đã nhập, ứng dụng sẽ tạo ra một mã QR để người dùng có thể vào bãi đậu. Tại phần giao diện chính của ứng dụng sẽ xuất hiện các thông báo đẩy để nhắc nhở người dùng đậu xe vào đúng vị trí đã chọn khi đặt chỗ. Người dùng sẽ thực hiện thanh toán trực tiếp trên ứng dụng dựa trên số giờ đã đậu xe trong bãi. Khi quá trình thanh toán thành công, người dùng sẽ nhận được mã QR để quét ra khỏi bãi đậu xe.

Tiêu chí của ứng dụng được thiết kế giúp người dùng theo dõi được tình trạng xe đậu trong bãi thực tế, giúp giảm thiểu tối đa thời gian chờ đợi phải có tốc độ phản hồi tối đa là 300 ms, độ trễ mạng khi nhận dữ liệu về tình trạng chỗ tối đa là 1 giây trong điều kiện mạng ổn định và tối đa 3 giây trong trường hợp mạng kém ổn định để đảm bảo trải nghiệm của người dùng. Đối với hệ thống, các cảm biến phải có độ trễ không quá 10 ms để có thể phát hiện xe, các dữ liệu được truyền và nhận từ web server phải dưới 100 ms trong điều kiện kết nối mạng ổn định. Thời gian quét và giải mã QR của máy quét phải dưới 300 ms, đáp ứng phạm vi quét từ 10 cm đến 60 cm với tỷ lệ lỗi dưới 0.01%. Đối với thanh chắn đặt tại lối ra vào của bãi đậu phải có tốc độ phản hồi

dưới 1 giây với góc xoay tối đa là 180 độ và đảm bảo độ lệch không quá ±1 độ để đảm bảo cổng được mở đóng chính xác theo yêu cầu.

1.3 Ý tưởng và phương pháp thực hiện

1.3.1 Ý tưởng

Một bản đồ sẽ được tích hợp vào ứng dụng dựa trên việc tạo API key trên Google Cloud console và sử dụng nó để kết nối ứng dụng thiết kế với Google Maps. Các Marker sẽ được tạo dựa trên vĩ độ và kinh độ tại các vị trí ta mong muốn đặt bãi đậu xe, người dùng sẽ dựa vào đó để xác định được vị trí bãi đậu xe gần nhất và lựa chọn để kiểm tra tình trạng bãi. Giải thuật chống trùng lặp sẽ được sử dụng dựa trên việc so sánh vị trí đậu và thời gian đặt chỗ với người đặt chỗ trước đó để hạn chế việc ấn đặt chỗ nhiều lần, mang lại tính công bằng cho người dùng. Dữ liệu về sự thay đổi trạng thái của các cảm biến hồng ngoại được lấy từ web server sẽ giúp cho ứng dụng biết chính xác vị trí xe rời khỏi bãi để hiển thị yêu cầu người dùng thanh toán phí đậu xe tại vị trí đó và được so sánh với vị trí đậu mà người dùng đã đặt chỗ để thông báo vị trí đậu xe của người dùng đã chính xác hay chưa. Thông tin đặt chỗ do người dùng cung cấp sẽ được mã hoá thành một mã QR để người dùng quét vào cổng thông qua việc thêm thư viện zxing vào ứng dụng và được lưu trữ thành một chuỗi byte trong một Java Class riêng biệt thông qua intent với kiểu truyền tuần tự (Serializable). Dữ liệu được lưu trữ từ Java Class được chuyển qua một Activity mới sau khi người dùng thực hiện thanh toán để tạo lại mã QR được người dùng sử dụng để quét ra khỏi bãi xe mà không cần phải nhập thông tin một lần nữa.

1.3.2 Phương pháp thực hiện

Đối với phần cứng, ESP32 sẽ là vi điều khiển chính được sử dụng cho hệ thống, được lập trình dựa trên phần mềm Arduino, sử dụng kết nối WiFi trên vi điều khiển để gửi các dữ liệu về vị trí chỗ đậu đã chọn khi người dùng quét mã QR, tình trạng và thời gian xe đậu trong bãi do các cảm biến hồng ngoại thu thập lên web server để lưu trữ. Cảm biến hồng ngoại được sử dụng và lập trình ở mức thấp và chuyển thành mức cao

dùng để phát hiện và xác định vị trí các xe trong bãi đậu cũng như phát hiện xe ra vào bãi. Servo được sử dụng để mô phỏng barrier và được lập trình với góc quay 90 độ cho việc mở cổng và quay về góc 0 độ để thực hiện đóng cổng thông qua việc gửi tín hiệu xung từ vi điều khiển đến servo dựa trên kết quả quét mã QR và cảm biến hồng ngoại. Bãi xe sẽ được trang bị thêm module GM65 chịu trách nhiệm giải thuật và xử lý hình ảnh về các hình ảnh mã QR được quét khi người dùng ra vào cổng để gửi dữ liệu về cho vi điều khiển thực hiện so sánh. Dữ liệu quét được nếu đúng với dữ liệu được lập trình thì hệ thống sẽ thực hiện đóng mở cổng cho xe ra vào.

Đối với phần mềm, một ứng dụng được thiết kế với ngôn ngữ Java dựa trên phần mềm Android Studio giúp cho người dùng có nhu cầu đậu xe biết được thông tin và vị trí bãi đậu chính xác theo thời gian thực, đặt chỗ trực tiếp dễ dàng và nhanh chóng. Dữ liệu về đặt chỗ của người dùng sẽ được mã hoá dưới dạng nhị phân và chuyển thành một mã QR để người dùng thực hiện quét vào cổng. Mã QR này sẽ được lưu trữ lại theo từng tài khoản của người dùng để thực hiện sử dụng lại khi người dùng muốn quét ra khỏi cổng. Ở giao diện thanh toán, cảm biến hồng ngoại phát hiện xe đậu trong bãi khi có sự thay đổi từ mức cao về mức thấp thì dữ liệu sẽ được vi điều khiển truyền về cơ sở dữ liệu và được ứng dụng truy xuất để hiển thị cho người dùng về thời gian xe đã đậu trong bãi để thực hiện tính phí đậu xe. Người dùng khi đã thanh toán thành công phí đậu xe thì sẽ được cung cấp lại mã QR đã được lưu trữ trước đó để ra khỏi bãi đậu xe.

Hệ thống sẽ sử dụng Firebase là cơ sở dữ liệu chính dùng để lưu trữ dữ liệu theo thời gian thực về tình trạng bãi xe và thời gian đậu xe do ESP32 gửi lên và được ứng dụng lấy về để cung cấp thông tin đến cho người sử dụng. Ngoài ra, hệ thống cũng thực hiện quản lý người dùng và thông tin đặt chỗ.

CHƯƠNG 2. CƠ SỞ LÝ THUYẾT

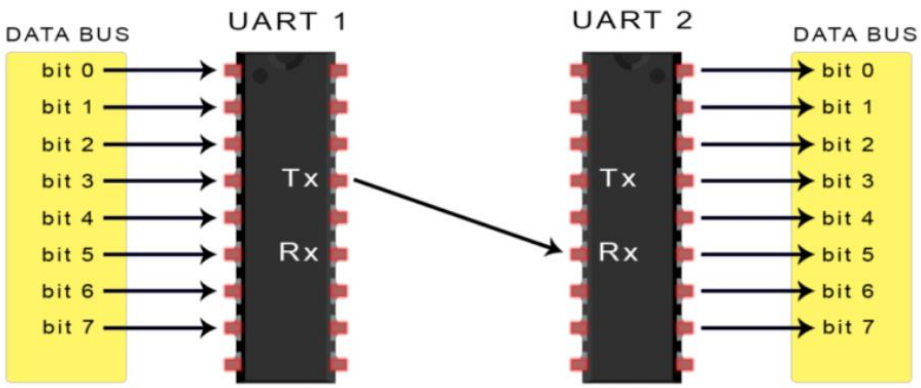
2.1 Tổng quan về UART

UART (Universal Asynchronous Receiver - Transmitter – Bộ truyền nhận dữ liệu không đồng bộ) là một giao thức truyền thông phần cứng dùng giao tiếp nối tiếp không đồng bộ và có thể cấu hình được tốc độ. Giao thức UART là một giao thức đơn giản và phổ biến, bao gồm hai đường truyền dữ liệu độc lập là TX (truyền) và RX (nhận). Dữ liệu được truyền và nhận qua các đường truyền này dưới dạng các khung dữ liệu (data frame) có cấu trúc chuẩn, với một bit bắt đầu (start bit), một số bit dữ liệu (data bits), một bit kiểm tra chẵn lẻ (parity bit) và một hoặc nhiều bit dừng (stop bit). Thông thường, tốc độ truyền của UART được đặt ở một số chuẩn, chẳng hạn như 9600, 19200, 38400, 57600, 115200 baud và các tốc độ khác. Tốc độ truyền này định nghĩa số lượng bit được truyền qua mỗi giây. Các tốc độ truyền khác nhau thường được sử dụng tùy thuộc vào ứng dụng và hệ thống sử dụng.

Về nguyên lý hoạt động, UART truyền dữ liệu nối tiếp, theo 1 trong 3 chế độ:

Bảng 2-1: Nguyên lý hoạt động của UART

Chế độ	Nguyên lý hoạt động
Simplex	Chỉ tiến hành giao tiếp một chiều
Half duplex	Dữ liệu sẽ đi theo một hướng tại 1 thời điểm
Full duplex	Thực hiện giao tiếp đồng thời đến và đi từ mỗi Master và Slave



Hình 2-1: Nguyên lý hoạt động của UART

Chân Tx (truyền) của một chip sẽ kết nối trực tiếp với chân Rx (nhận) của chip khác và ngược lại. Quá trình truyền dữ liệu thường sẽ diễn ra ở 3.3V hoặc 5V. UART là một giao thức giao tiếp giữa một Master và một Slave. Trong đó 1 thiết bị được thiết lập để tiến hành giao tiếp với chỉ duy nhất 1 thiết bị khác.

Dữ liệu truyền đến và đi từ UART song song với thiết bị điều khiển. Khi tín hiệu gửi trên chân Tx (truyền), bộ giao tiếp UART đầu tiên sẽ dịch thông tin song song này thành dạng nối tiếp và sau đó truyền tới thiết bị nhận. Chân Rx (nhận) của UART thứ 2 sẽ biến đổi nó trở lại thành dạng song song để giao tiếp với các thiết bị điều khiển. Dữ liệu truyền qua UART sẽ đóng thành các gói (packet). Mỗi gói dữ liệu chứa 1 bit bắt đầu, 5 – 9 bit dữ liệu (tùy thuộc vào bộ UART), 1 bit chẵn lẻ tùy chọn và 1 bit hoặc 2 bit dừng. Quá trình truyền dữ liệu UART sẽ diễn ra dưới dạng các gói dữ liệu này, bắt đầu bằng 1 bit bắt đầu, đường mức cao được kéo dần xuống thấp. Sau bit bắt đầu là 5 – 9 bit dữ liệu truyền trong khung dữ liệu của gói, theo sau là bit chẵn lẻ tùy chọn để nhằm xác minh việc truyền dữ liệu thích hợp. Sau cùng, 1 hoặc nhiều bit dừng sẽ được truyền ở nơi đường đặt tại mức cao. Vậy là sẽ kết thúc việc truyền đi một gói dữ liệu.

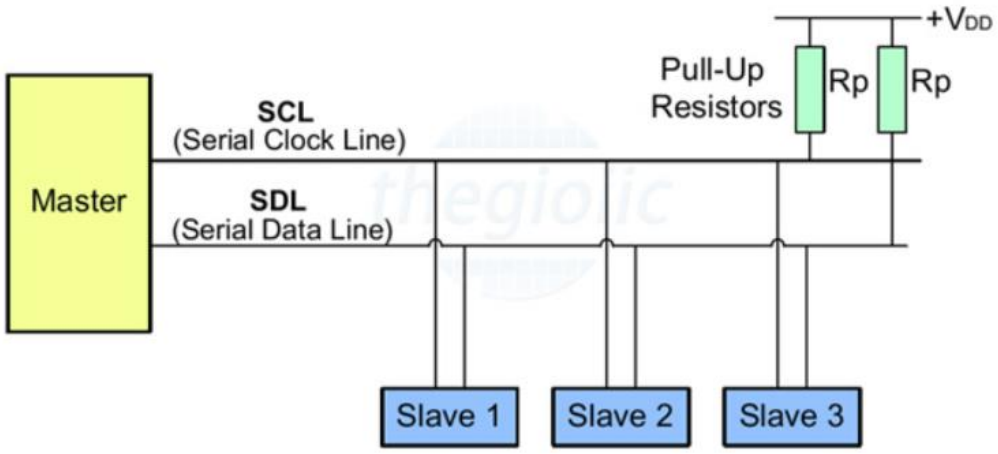
Ưu điểm của giao tiếp UART là chỉ cần dùng 2 dây truyền dữ liệu, không cần dùng tín hiệu clock, có 2 bit chẵn lẻ nên có thể kiểm tra lỗi dễ dàng, cấu trúc gói dữ liệu có thể thay đổi được miễn là cả 2 bên đều được thiết lập để giao tiếp với nhau, phương pháp giao tiếp UART có nhiều tài liệu hướng dẫn và cũng là bộ truyền dữ liệu đang được sử dụng rộng rãi hiện nay. Tuy nhiên, nó cũng tồn tại không ít mặt hạn chế như kích thước của khung dữ liệu giới hạn tối đa là 9 bit, khá nhỏ so với nhu cầu sử dụng, không được hỗ trợ nhiều hệ thống Master và Slave, tốc độ truyền của mỗi giao tiếp UART phải nằm trong khoảng 10% của nhau.

Giao tiếp UART có nhiều ứng dụng cơ bản trong lĩnh vực điện tử và tự động hóa. Nó thường được sử dụng để kết nối các thiết bị điện tử với máy tính như vi điều khiển, cảm biến, màn hình hiển thị, máy in và các thiết bị khác. Khi kết nối với máy tính, các thiết bị này có thể gửi và nhận dữ liệu qua cổng UART. Ngoài ra, UART còn được sử dụng để giao tiếp giữa các vi điều khiển, cho phép chúng truyền và nhận dữ

liệu để thực hiện các tác vụ như điều khiển động cơ, đo lường và kiểm soát các thông số cũng như điều khiển các thiết bị khác. Trong giao tiếp không dây, giao thức UART có thể được sử dụng để kết nối các thiết bị Bluetooth hoặc WiFi, nơi dữ liệu được truyền qua sóng radio hoặc sóng vô tuyến. Đối với điều khiển robot, các bộ điều khiển và module điều khiển robot thường sử dụng giao thức UART để gửi và nhận dữ liệu từ nhau, giúp thực hiện các thao tác điều khiển. Cuối cùng, trong các hệ thống đo lường, UART được sử dụng để kết nối các thiết bị đo lường với các thiết bị khác, chẳng hạn như máy tính hoặc các thiết bị nhúng. Các thiết bị đo lường có thể gửi dữ liệu về các thông số đo được thông qua cổng UART và thiết bị nhận có thể hiển thị hoặc xử lý dữ liệu này.

2.2 Chuẩn giao tiếp I2C

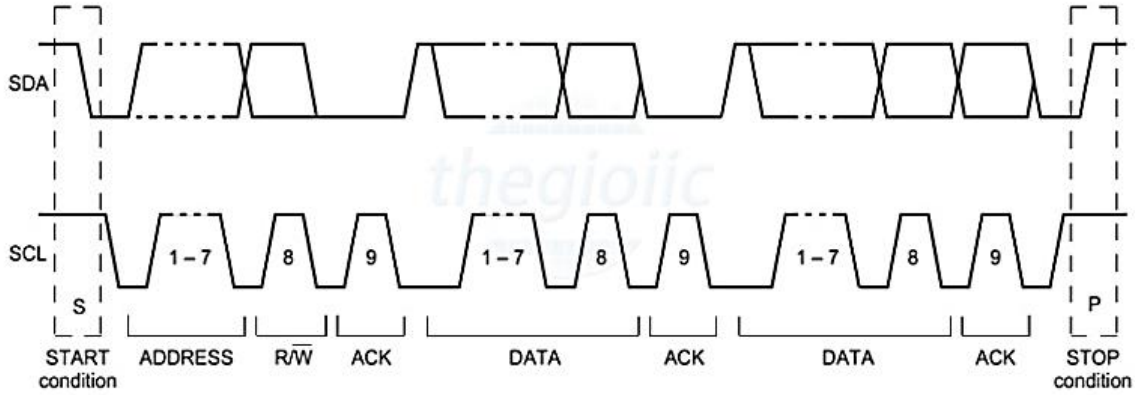
I2C hay IIC (Inter – Integrated Circuit) là 1 giao thức giao tiếp nối tiếp đồng bộ được phát triển bởi Philips Semiconductors, sử dụng để truyền nhận dữ liệu giữa các IC với nhau chỉ sử dụng hai đường truyền tín hiệu. I2C kết hợp các tính năng tốt nhất của SPI và UART. I2C có thể kết nối nhiều Slave với một Master duy nhất (như SPI) và có thể có nhiều Master điều khiển một hoặc nhiều Slave. Điều này thực sự cần thiết khi muốn có nhiều hơn một vi điều khiển ghi dữ liệu vào một thẻ nhớ duy nhất hoặc hiển thị văn bản trên một màn hình LCD.



Hình 2-2: Giao tiếp I2C

Giống như giao tiếp UART, I2C chỉ sử dụng hai dây để truyền dữ liệu giữa các thiết bị bao gồm: SDA (Serial Data) - đường truyền cho Master và Slave để gửi và nhận dữ liệu SCL (Serial Clock) - đường mang tín hiệu xung nhịp. Các bit dữ liệu sẽ được truyền từng bit một dọc theo một đường duy nhất (SDA) theo các khoảng thời gian đều đặn được thiết lập bởi 1 tín hiệu đồng hồ (SCL).

Giao tiếp I2C bao gồm quá trình truyền nhận dữ liệu giữa các thiết bị chủ tớ, hay Master - Slave. Thiết bị Master là 1 vi điều khiển, nó có nhiệm vụ điều khiển đường tín hiệu SCL và gửi nhận dữ liệu hay lệnh thông qua đường SDA đến các thiết bị khác. Các thiết bị nhận các dữ liệu lệnh và tín hiệu từ thiết bị Master được gọi là các thiết bị Slave. Các thiết bị Slave thường là các IC hoặc thậm chí là vi điều khiển, màn hình, cảm biến và nhiều thiết bị ngoại vi khác có khả năng giao tiếp qua I2C.



Hình 2-3: Khung dữ liệu của I2C

Master và Slave được kết nối với nhau bằng hai đường bus SCL và SDA đều hoạt động ở chế độ Open Drain, nghĩa là bất cứ thiết bị nào kết nối với mạng I2C này cũng chỉ có thể kéo 2 đường bus này xuống mức thấp (LOW) nhưng lại không thể kéo được lên mức cao. Một điện trở (từ 1 – 4.7 kΩ) được giữ mặc định ở mức cao để tránh trường hợp bus vừa bị 1 thiết bị kéo lên mức cao vừa bị 1 thiết bị khác kéo xuống mức thấp gây hiện tượng ngắn mạch.

Với I2C thì dữ liệu được truyền trong các tin nhắn, tin nhắn được chia thành các khung dữ liệu. Mỗi tin nhắn có một khung địa chỉ chứa địa chỉ nhị phân của địa chỉ Slave và một hoặc nhiều khung dữ liệu chứa dữ liệu đang được truyền. Thông điệp

cũng bao gồm điều kiện khởi động và điều kiện dừng, các bit đọc/ghi và các bit ACK/NACK giữa mỗi khung dữ liệu:

Bảng 2-2: Các thành phần trong khung dữ liệu

Thành phần	Hoạt động
Điều kiện khởi động	Đường SDA chuyển từ mức điện áp cao xuống mức điện áp thấp trước khi đường SCL chuyển từ mức cao xuống mức thấp.
Điều kiện dừng	Đường SDA chuyển từ mức điện áp thấp sang mức điện áp cao sau khi đường SCL chuyển từ mức thấp lên mức cao.
Bit địa chỉ	Thông thường quá trình truyền nhận sẽ diễn ra với rất nhiều thiết bị, IC với nhau. Do đó để phân biệt các thiết bị này, chúng sẽ được gắn 1 địa chỉ vật lý 7 bit cố định.
Bit đọc/ghi	Bit này dùng để xác định quá trình là truyền hay nhận dữ liệu từ thiết bị Master. Nếu Master gửi dữ liệu đi thì ứng với bit này bằng ‘0’ và ngược lại, nhận dữ liệu khi bit này bằng ‘1’.
Bit ACK/NACK	Dùng để so sánh bit địa chỉ vật lý của thiết bị so với địa chỉ được gửi tới. Nếu trùng thì Slave sẽ được đặt bằng ‘0’ và ngược lại, nếu không thì mặc định bằng ‘1’.
Bit dữ liệu	Gồm 8 bit và được thiết lập bởi thiết bị gửi truyền đến thiết bị nhân. Sau khi các bit này được gửi đi, lập tức 1 bit ACK/NACK được gửi ngay theo sau để xác nhận rằng thiết bị nhận đã nhận được dữ liệu thành công hay chưa. Nếu nhận thành công thì bit ACK/NACK được set bằng ‘0’ và ngược lại.

Khi bắt đầu, Master sẽ gửi đi 1 xung Start bằng cách kéo lần lượt các đường SDA, SCL từ mức 1 xuống 0. Tiếp theo đó, Master gửi đi 7 bit địa chỉ tới các Slave cùng với bit Read/Write. Slave sẽ so sánh địa chỉ vừa được gửi tới. Nếu trùng khớp, Slave

sẽ xác nhận bằng cách kéo đường SDA xuống 0 và set bit ACK/NACK bằng '0'. Nếu không trùng khớp thì SDA và bit ACK/NACK đều mặc định bằng '1'. Thiết bị Master sẽ gửi hoặc nhận khung bit dữ liệu. Nếu Master gửi đến Slave thì bit Read/Write ở mức 0. Ngược lại nếu nhận thì bit này ở mức 1. Nếu như khung dữ liệu đã được truyền đi thành công, bit ACK/NACK được set thành mức 0 để báo hiệu cho Master tiếp tục. Sau khi tất cả dữ liệu đã được gửi đến Slave thành công, Master sẽ phát 1 tín hiệu Stop để báo cho các Slave biết quá trình truyền đã kết thúc bằng các chuyển lần lượt SCL, SDA từ mức 0 lên mức 1.

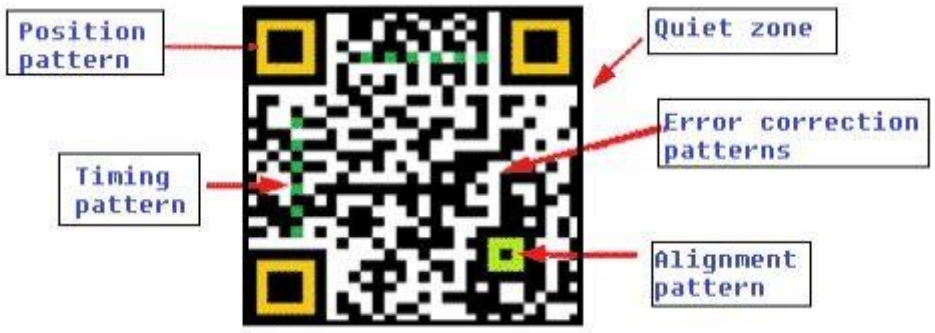
I2C có 2 chế độ hoạt động bao gồm: chế độ chuẩn (standard mode) với tốc độ 100 kBit/s và chế độ tốc độ thấp (low speed mode) với tốc độ 10 kBit/s. Khác với giao tiếp SPI chỉ có thể có 1 Master, giao tiếp I2C cho phép chế độ truyền nhận dữ liệu giữa nhiều thiết bị Master khác nhau với thiết bị Slave. Tuy nhiên quá trình này có hơi phức tạp vì thiết bị Slave có thể nhận 1 lúc nhiều khung dữ liệu từ các thiết bị Master khác nhau, điều đó đôi khi dẫn đến xung đột hoặc sai sót dữ liệu nhận được. Để tránh điều đó, khi làm việc ở chế độ này, mỗi thiết bị Master cần phát hiện xem đường SDA đang ở trạng thái nào. Nếu SDA ở mức 0 nghĩa là đang có 1 thiết bị Master khác đang có quyền điều khiển và phải chờ đến khi truyền xong. Ngược lại nếu SDA ở mức 1 nghĩa là đường truyền SDA đã an toàn và có sử dụng.

Chuẩn giao tiếp I2C có ưu điểm là chỉ sử dụng hai dây, hỗ trợ nhiều Master và nhiều Slave, bit ACK/NACK xác nhận mỗi khung được chuyển thành công, phần cứng ít phức tạp hơn so với UART, giao thức nổi tiếng và được sử dụng rộng rãi. Bên cạnh đó, nó cũng có một vài nhược điểm như tốc độ truyền dữ liệu chậm hơn SPI, kích thước của khung dữ liệu bị giới hạn ở 8 bit, cần phần cứng phức tạp hơn để triển khai so với SPI.

2.3 QR code

QR Code là một ma trận được tạo bởi hỗn hợp các hình vuông và các chấm đen, trắng. QR code là mã vạch 2 chiều (2D) tức là có thể đọc theo 2 chiều ngang và chiều dọc bên trong chứa các thông tin như địa chỉ trang web, thông tin liên hệ, đoạn văn bản,…

QR code gồm có 5 cấu tạo chính, mỗi phần đóng vai trò quan trọng trong việc lưu trữ và đọc thông tin. Mã QR sẽ gồm nhiều ô (cell) màu trắng và đen mang giá trị nhị phân là 0 và 1, tập hợp của các ô này chính là phần thông tin được lưu trữ và mã hoá trong mã QR. Ở bốn góc của mã QR có các ô vuông gọi là hoa văn định vị, chúng giúp camera xác định phạm vi của mã QR và đọc chính xác thông tin ngay cả khi mã bị biến dạng. Đồng thời, hoa văn này cũng giúp phân cách mã với các ký tự hoặc hình vẽ xung quanh. Vùng chứa các ô đen trắng được sắp xếp xen kẽ được gọi là timing pattern dùng để xác định tọa độ của mã QR, giúp mã được định vị chính xác. Ở bên phải của mã QR, có một ô vuông nhỏ bên trong được gọi là alignment pattern giúp điều chỉnh các chênh lệch xảy ra khi quét mã QR trong trường hợp mã bị lệch. Xung quanh các hoa văn định vị là phần chứa thông tin format, quyết định mức độ sửa lỗi của mã QR code. Mã QR cũng sử dụng cơ chế "mask" để đảm bảo sự cân bằng giữa các ô trắng và đen, tránh tình trạng mã chứa toàn màu đen hoặc trắng. Điều này giúp thông tin lưu trữ trong mã QR được bảo toàn, đồng thời giữ sự cân bằng về màu sắc để mã hoạt động chính xác.



Hình 2-4: Cấu tạo QR code

Khả năng sửa chữa lỗi của QR code được chia làm 4 mức độ: L, M, Q, H. Mức độ sửa chữa lỗi càng cao thì khả năng kháng lại lỗi rách, hỏng của QR code càng cao. Trong các trường hợp thông thường, QR code với mức độ sửa chữa lỗi M được sử dụng. Trong các môi trường QR code dễ bị bẩn, rách như công xưởng, công trường, code level Q hoặc H được sử dụng.

Bảng 2-3: Mức độ sửa chữa lỗi của QR code

Mức độ sửa chữa lỗi của QR code	Độ khôi phục	Ứng dụng
Level L	khoảng 7%	Sử dụng trong môi trường ít bị bẩn
Level M	khoảng 15%	Sử dụng trong môi trường thông thường
Level Q	khoảng 25%	Sử dụng trong môi trường dễ bị bẩn như xưởng sản xuất
Level H	khoảng 30%	Sử dụng trong môi trường dễ bị bẩn như xưởng sản xuất

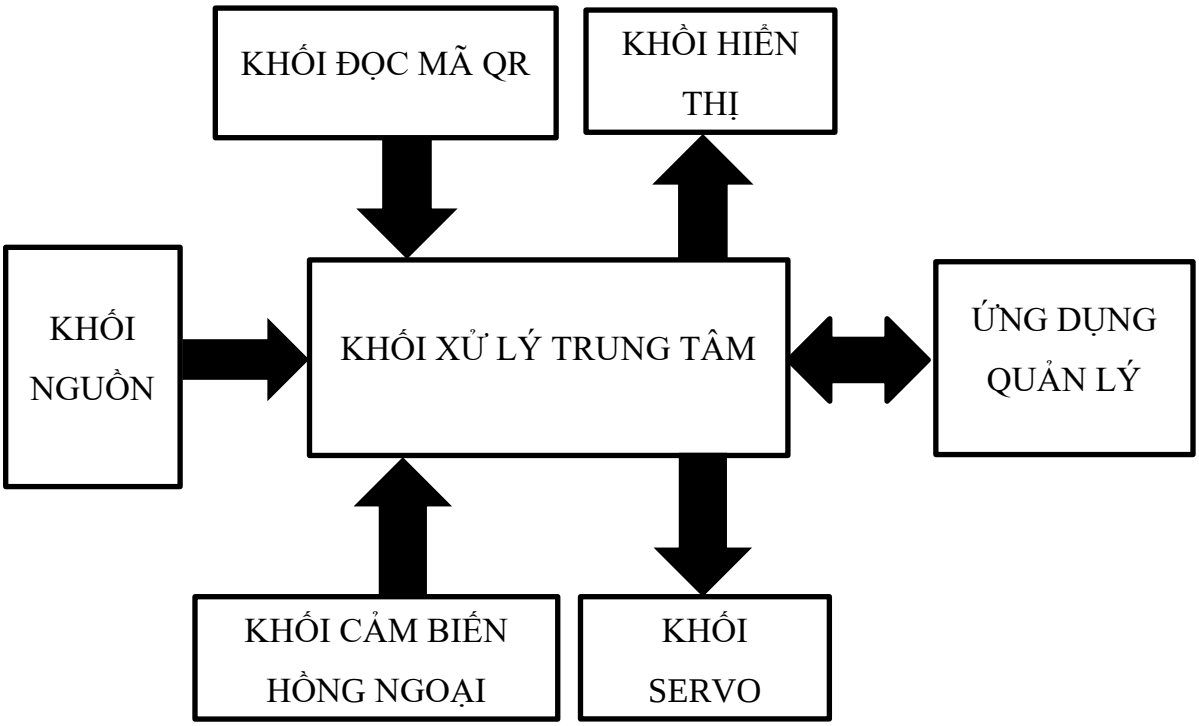
Quá trình tạo ra một mã QR code được thực hiện gồm mã hóa dưới dạng nhị phân (chuỗi các số 0 và 1) cho thông tin mà bạn muốn lưu trữ trong mã QR. Thông tin này có thể là văn bản, URL, số điện thoại hay bất kỳ dữ liệu nào có thể được chuyển đổi sang định dạng nhị phân. Dữ liệu được chia thành nhiều khối nhỏ, mỗi khối sẽ chứa một phần của chuỗi nhị phân, giúp cho việc xử lý dữ liệu dễ dàng và chính xác hơn khi quét mã. Để đảm bảo rằng mã QR vẫn có thể được đọc ngay cả khi bị hư hỏng hay mờ, thuật toán sửa lỗi Reed - Solomon sẽ được áp dụng. Mức độ sửa lỗi này có thể tùy chỉnh (L, M, Q, H) với mức cao nhất cho phép mã QR vẫn hoạt động ngay cả khi lên đến 30% diện tích bị hỏng. Các hoa văn định vị (ở 3 góc mã QR) và thông tin định vị timing pattern (các ô đen trắng xen kẽ), alignment pattern (ô vuông nhỏ ở bên phải mã QR) được chèn vào mã. Những hoa văn này giúp camera hoặc thiết bị đọc mã QR dễ dàng xác định kích thước, hướng và độ thẳng của mã QR. Để đảm bảo mã QR không quá mất cân bằng giữa các ô đen và trắng, một lớp mask sẽ được áp dụng. Mask giúp phân bố đều các ô đen trắng, tránh trường hợp mã QR chỉ chứa một màu nhiều hơn, gây khó khăn khi quét. Các ô trắng và đen sẽ được sắp xếp vào một lưới hình vuông, từ đó tạo ra hình ảnh mã QR cuối cùng. Mỗi ô đen hoặc trắng đại diện cho một phần của chuỗi nhị phân đã mã hóa ban đầu. Mã QR hoàn chỉnh sẽ được xuất ra dưới dạng hình ảnh có thể in hoặc hiển thị trên các phương tiện kỹ thuật số. Người

dùng có thể quét mã QR này bằng ứng dụng hoặc thiết bị đọc mã để giải mã và truy cập thông tin chứa trong đó.

Khi sử dụng camera hoặc thiết bị quét mã QR, nó sẽ chụp lại hình ảnh của mã QR. Phần mềm giải mã sẽ sử dụng hình ảnh này để bắt đầu quá trình phân tích và nhận diện. Phần mềm xác định vị trí và hướng của mã QR dựa trên các hoa văn định vị (các ô vuông lớn ở 3 góc mã QR). Các hoa văn này giúp phần mềm biết được đâu là góc trên, dưới, trái, phải, và xác định được kích thước mã QR. Sau khi xác định được hướng, phần mềm sẽ tìm timing pattern (các ô đen trắng xen kẽ) và alignment pattern (ô vuông nhỏ ở bên phải mã QR). Các hoa văn này giúp phần mềm định vị chính xác các ô dữ liệu trong mã, ngay cả khi mã bị lệch hoặc méo. Mã QR được thiết kế với một lớp mask để phân bố đều các ô đen và trắng. Khi giải mã, phần mềm sẽ loại bỏ lớp mask để lấy lại dữ liệu gốc. Thông tin về cách áp dụng mask được lưu trữ trong vùng "thông tin format" của mã QR. Các ô đen và trắng đại diện cho các bit nhị phân (0 và 1) được đọc tuần tự theo thứ tự xác định. Các chuỗi nhị phân này chứa thông tin đã được mã hóa chẳng hạn như văn bản, URL hoặc số điện thoại. Mã QR có tích hợp cơ chế sửa lỗi theo thuật toán Reed - Solomon. Nếu một phần của mã QR bị hỏng hoặc mờ, phần mềm sẽ sử dụng thông tin từ các khối sửa lỗi để khôi phục lại dữ liệu bị mất. Tùy thuộc vào mức độ sửa lỗi được thiết lập, mã QR có thể chịu được mức độ hư hỏng từ 7% đến 30%. Sau khi phần mềm đã giải mã chuỗi nhị phân và sửa lỗi, dữ liệu này sẽ được chuyển đổi về dạng thông tin gốc chẳng hạn như văn bản hoặc đường dẫn web. Thông tin được giải mã từ mã QR sẽ được hiển thị trên thiết bị của bạn. Nếu mã QR chứa một URL, thiết bị sẽ tự động mở trang web đó hoặc nếu chứa văn bản thì văn bản đó sẽ được hiển thị.

CHƯƠNG 3. THIẾT KẾ BÃI ĐẬU XE THÔNG MINH

3.1 Thiết kế phần cứng

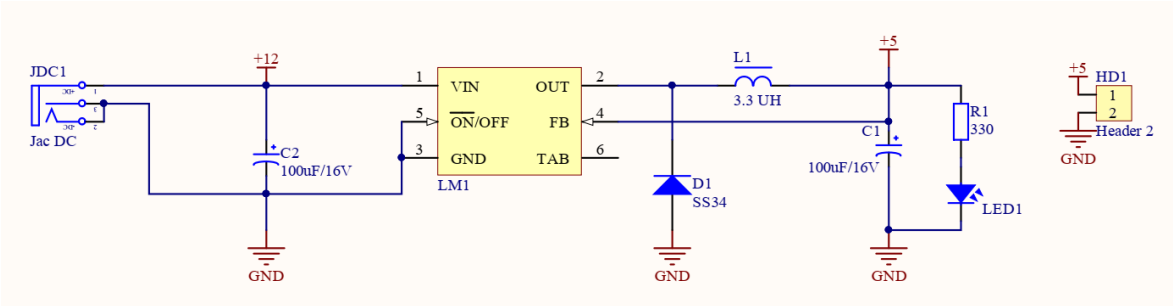


Hình 3-1: Sơ đồ khối hệ thống

Hệ thống bãi đậu xe thông minh bao gồm 7 khối, mỗi khối đảm nhận một nhiệm vụ riêng biệt để đảm bảo hệ thống hoạt động một cách hiệu quả. Khối nguồn là thành phần chịu trách nhiệm cung cấp nguồn điện cho toàn bộ hoạt động của hệ thống, đảm bảo các linh kiện đều hoạt động ổn định. Khối xử lý trung tâm đóng vai trò cốt lõi, nhận tín hiệu từ các thiết bị ngoại vi, xử lý và điều khiển toàn bộ hệ thống. Ngoài ra, khối này còn đảm nhiệm việc thu thập và gửi dữ liệu về lượng xe ra vào lên cơ sở dữ liệu để quản lý và giám sát. Khối đọc mã QR được sử dụng để quét mã QR từ ứng dụng thiết kế. Thông tin đọc được từ mã QR sẽ được gửi về vi điều khiển để phân tích và xử lý, đảm bảo quyền ra vào hợp lệ của xe. Khối cảm biến hồng ngoại có nhiệm vụ phát hiện xe ra vào, tính toán số lượng xe và truyền tín hiệu về cho vi điều khiển để kiểm soát quá trình ra vào của xe cũng như tình trạng bãi xe. Khối hiển thị giúp hiển thị thông tin quan trọng như số lượng xe hiện có trong bãi, trạng thái hoạt

động của hệ thống và yêu cầu quét mã QR khi có xe ra hoặc vào. Khối Servo điều khiển hoạt động đóng mở cổng của bãi đậu xe, cho phép xe ra vào sau khi đã xác nhận qua hệ thống. Ứng dụng quản lý hoạt động như một giao diện người dùng, nhận và quản lý dữ liệu từ cơ sở dữ liệu mà khối xử lý trung tâm gửi lên. Ứng dụng này theo dõi lượng xe có trong bãi, quản lý thông tin đặt chỗ của người dùng, đảm bảo quá trình điều phối xe trong bãi hiệu quả và chính xác.

3.2 Khối nguồn



Hình 3-2: Khối nguồn

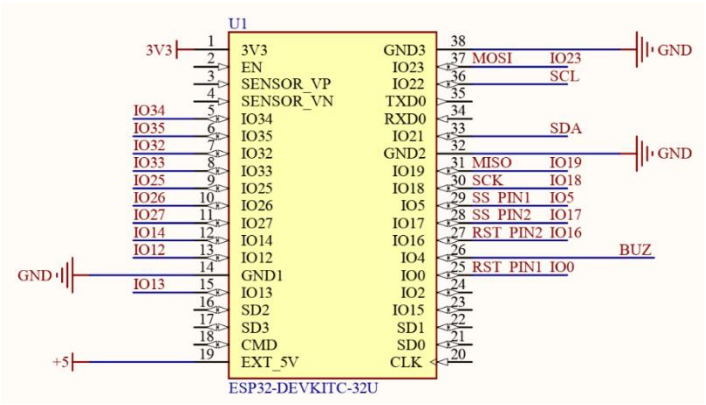
Khối nguồn trong sơ đồ nguyên lý này là một mạch buck converter (bộ chuyển đổi hạ áp) được thiết kế để chuyển đổi điện áp cung cấp từ Adapter 12 V xuống 5 V nhằm cung cấp điện cho toàn bộ hoạt động của hệ thống. IC được lựa chọn sử dụng trong mạch hạ áp này chính là LM2576 vì có thể hoạt động ở điện áp đầu vào từ 4.5 V đến 40 V và điện áp đầu ra có thể điều chỉnh hoặc cố định ở các mức 3.3 V, 5 V, 12 V và 15V. IC LM2576 rất phù hợp cho các ứng dụng cần ổn định điện áp như hệ thống bãi đậu xe bởi nó có thể bảo vệ quá dòng, quá nhiệt và quá áp. Cuộn cảm giúp lưu trữ năng lượng dưới dạng từ trường và giúp làm mịn dòng điện đầu ra được chọn với giá trị 3.3 µH dựa trên tần số chuyển mạch của IC LM2576 là 52 kHz và dòng điện đầu ra khoảng 1 A với độ gợn dòng qua cuộn cảm là 30% của dòng điện đầu ra cho mạch buck converter. Giá trị cuộn cảm lý tưởng sẽ vào khoảng 2.24 µH nên việc chọn cuộn cảm 3.3 µH là hợp lý để đáp ứng yêu cầu điện áp và dòng điện trong mạch. Diode Schottky có điện áp ngược chịu đựng khoảng 40 V và dòng tối đa 3 A, điện áp rơi thấp hơn (~0.3 V) so với các diode thông thường, giúp cải thiện hiệu suất của mạch. Diode Schottky bảo vệ mạch khỏi các dòng ngược khi cuộn cảm phóng điện trong chu kỳ chuyển đổi. Tụ điện có nhiệm vụ lọc nhiễu, làm mịn điện áp và giảm gợn sóng

đầu ra. Tụ C2 lọc đầu vào giúp duy trì sự ổn định cho điện áp 12 V còn tụ C1 lọc đầu ra 5 V giúp loại bỏ các nhiễu hoặc dao động sau quá trình chuyển đổi. Các tụ có giá trị 100 µF để cung cấp khả năng lọc hiệu quả, đồng thời điện áp định mức của tụ (10V) được chọn lớn hơn mức điện áp đầu ra (5 V) để đảm bảo an toàn. Đèn LED được sử dụng để báo trạng thái nguồn kết hợp cùng với điện trở 330 Ω để giới hạn dòng điện cho LED. Khi hệ thống cấp điện thành công (5 V) thì đèn LED sẽ sáng, giúp người dùng biết mạch nguồn hoạt động bình thường. Domino được sử dụng trong cổng kết nối đầu ra giúp cho hệ thống bãi đậu xe thông minh có được sự ổn định trong việc cung cấp nguồn 5 V cho các thành phần khác của hệ thống.

Bảng 3-1: Tổng dòng điện toàn mạch

STT	Thiết bị	Số lượng	Điện áp	Dòng điện	Tổng dòng điện
1	ESP32	1	5V	~160mA	~160mA
2	GM65	1	5V	~120mA	~120mA
3	Servo SG90	2	5V	~450mA	~900mA
4	LCD 20x4	1	5V	~30mA	~30mA
5	Module cảm biến hồng ngoại MH-B	6	5V	~55mA	~330mA
Tổng dòng điện					**~1.5A**

3.3 Khối xử lý trung tâm

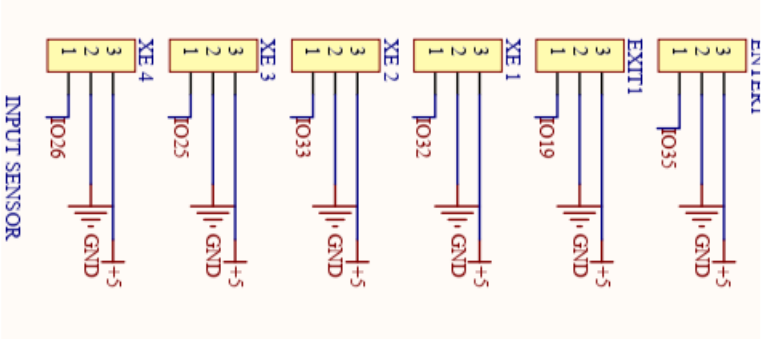


Hình 3-3: Khối xử lý trung tâm

Vi điều khiển được xem như là "trung tâm đầu não" của hệ thống, việc lựa chọn một vi điều khiển phù hợp cho hệ thống là rất quan trọng bởi nó cần phải đáp ứng đầy đủ các tiêu chí đặt ra, liên kết tất cả các linh kiện của hệ thống, nhận tín hiệu và xử lý tín hiệu đó một cách chính xác và đồng thời có thể gửi dữ liệu tình trạng bãi đậu lên cơ sở dữ liệu Firebase. Có rất nhiều loại vi điều khiển hiện nay phù hợp cho hệ thống có thể kể đến như Arduino UNO R3, Raspberry Pi, ESP32,… nhưng ở đây ESP32 là phù hợp nhất bởi giá thảnh rẻ, nhỏ gọn, có khả năng thu phát WiFi tốt theo chuẩn 802.11 b/g/n với băng tần 2.4 GHz. ESP32 còn được trang bị cổng nạp USB và hầu như hỗ trợ tất cả các loại giao tiếp cần thiết cho đề tài như giao tiếp I2C với màn hình LCD, giao tiếp UART với module quét mã QR GM65 và giao tiếp PWM với động cơ Servo. Nguồn Adapter 12 V qua mạch buck converter sẽ chuyển đổi điện áp từ 12 V xuống 5 V để cung cấp cho ESP32, quá trình này nhằm đảm bảo ESP32 nhận đúng mức điện áp 5 V để hoạt động và có thể phân phối điện áp cho các thiết bị khác mà không bị nóng hoặc hỏng hóc do quá điện áp. Các chân được bố trí trên ESP32 tuỳ thuộc vào các giao tiếp với linh kiện, chẳng hạn như đối với giao tiếp I2C cho việc hiển thị thông tin trên LCD, ta sẽ sử dụng chân 21 và 22 trên ESP32. Đối với giao tiếp UART, ta sẽ bố trí chân 16 và 17 để GM65 dễ dàng giao tiếp và không bị nhiễu tín hiệu. Đối với giao tiếp PWM trên servo, chúng ta sẽ bố trí chân 12 và 14 của ESP32 cho việc truyền tín hiệu từ ESP32 đến servo vì khả năng chịu dòng tốt và các chân này không bị ràng buộc bởi các chức năng khác quan trọng như SPI, I2C,… nên sẽ không gây xung đột với các thiết bị ngoại vi khác. Đối với các chân 35, 19, 32, 33, 25 và 26 được sử dụng để bố trí cho các cảm biến hồng ngoại bởi vì các chân này có khả năng đọc tín hiệu analog từ cảm biến hồng ngoại. Cảm biến hồng ngoại thường xuất ra tín hiệu analog dựa trên cường độ ánh sáng phản xạ từ vật thể và ESP32 có các chân hỗ trợ đọc ADC (Analog to Digital Converter) để chuyển đổi tín hiệu analog thành giá trị số mà vi điều khiển có thể xử lý. Ngoài ra, ở các chân được bố trí cho cảm biến hồng ngoại thường không được sử dụng cho các giao tiếp SPI, I2C hay UART nên sẽ tránh được xung đột với các thiết bị ngoại vi khác. Việc sử dụng các

chân ở nhiều vị trí khác nhau trên ESP32 giúp cân bằng tải và giảm thiểu nhiễu chéo (crosstalk) giữa các tín hiệu, đặc biệt là khi làm việc các cảm biến hồng ngoại.

3.4 Khối cảm biến hồng ngoại



Hình 3-4: Khối cảm biến hồng ngoại

Cảm biến được sử dụng trong việc phát hiện vật cản có thể ứng dụng trong việc phát hiện có xe ra vào bãi, biết được vị trí xe đang đậu cũng như kiểm soát được số lượng xe ra vào bãi. Khi lựa chọn thiết kế, ta thường nghĩ ngay đến việc sử dụng cảm biến hồng ngoại ngõ ra analog hoặc digital hoặc cảm biến siêu âm bởi ưu điểm ít nhiễu và độ chính xác tương đối cao. Tuy nhiên, trong đề tài này module LED thu phát hồng ngoại với ngõ ra digital sẽ ưu tiên lựa chọn bởi vì thuận tiện trong việc điều khiển và giá thành hợp lí. Mặc dù cảm biến này chưa chống nhiễu ánh sáng thật sự tốt nhưng về cơ bản vẫn đáp ứng được yêu cầu đặt ra. Sáu cảm biến hồng ngoại sẽ được sử dụng bao gồm 1 cảm biến sử dụng ở cửa vào được kết nối với chân 35 của ESP32, 1 cảm biến sử dụng ở cửa ra đươc kết nối với chân 19 của ESP32 và 4 cảm biến sử dụng cho việc xác định vị trí xe trong bãi kết nối với chân 32, 33, 25, 26 tương ứng với vị trí đậu từ 1 đến 4. Tất cả các cảm biến hồng ngoại đều sử dụng chung nguồn cấp +5 V và GND vì vậy việc bố trí như trong Hình 3-4 sẽ giúp tiết kiệm không gian và dây dẫn cũng như giảm thiểu sai sót khi đấu nối với một đường cấp nguồn duy nhất sẽ dễ dàng trong việc quản lý và kiểm tra. Ngoài ra, việc bố trí các cảm biến như trên giúp đảm bảo rằng tất cả các cảm biến hoạt động ở cùng một mức điện áp, giảm nguy cơ hư hỏng do điện áp không đồng đều hoặc sai lệch. Việc sử dụng các cảm biến hồng ngoại kết nối trực tiếp với ngõ ra digital trên ESP32 cũng giúp giảm thiểu nguy cơ gây nhiễu hoặc suy giảm tín hiệu.

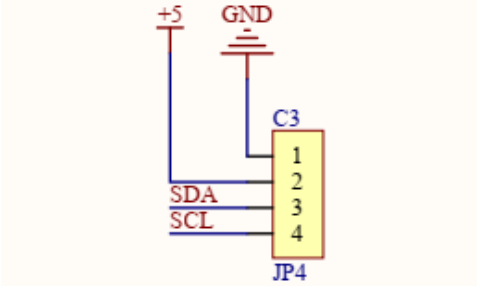
3.5 Khối Servo

Hình 3-5: Khối Servo

Để ứng dụng vào việc đóng mở barrier cho xe ra vào khi thực hiện thao tác quét mã QR, ta thường nghĩ ngay đến việc sử dụng khối động cơ Servo. Do hệ thống nhỏ chỉ dùng cho mô phỏng nên ta sẽ chọn động cơ servo SG90 với điện áp đầu vào 5 V đủ đáp ứng cho nhu cầu điều khiển và tối ưu kinh tế. Ta sẽ sử dụng 2 servo với 1 servo được dùng ở lối vào và 1 được dùng ở lối ra. Servo có 3 chân với 3 màu gồm vàng, đỏ, nâu tương ứng với chân điều khiển, chân nguồn 5 V và chân mass. Chân điều khiển được dùng để kết nối với chân số 14 chân số 12 của ESP32 ở lối ra và lối vào của hệ thống. ESP32 có khả năng tạo ra tín hiệu PWM nên rất hữu ích để điều khiển servo. PWM là tín hiệu dạng sóng với chu kỳ làm việc (duty cycle) của sóng sẽ quyết định góc quay của servo. Thời gian xung từ 1 ms đến 2 ms sẽ tương ứng với góc quay từ 0 độ đến 180 độ. Khi servo nhận được tín hiệu từ ESP32 với thời gian xung là 1.5 ms cho góc quay 90 độ sẽ thực hiện việc đóng mở servo tương tự như barrier cho xe ra/vào. Servo được thiết kế cấp nguồn riêng từ nguồn ngoài thay vì từ ESP32 vì các servo thường tiêu thụ dòng điện khá lớn có thể gây quá tải cho nguồn điện của ESP32 nếu cấp trực tiếp. Việc sử dụng nguồn riêng cho servo giúp đảm bảo sự ổn định cho cả servo và ESP32.

3.6 Khối hiển thị

Để hiển thị thông tin về tình trạng bãi đậu giúp cho người điều khiển phương tiện cũng như người đậu xe dễ dàng quan sát xem liệu bãi đậu còn chỗ trống hay đã hết chỗ thì ta thường sử dụng LCD, LED 7 đoạn và LED ma trận.



Hình 3-6: Khối hiển thị

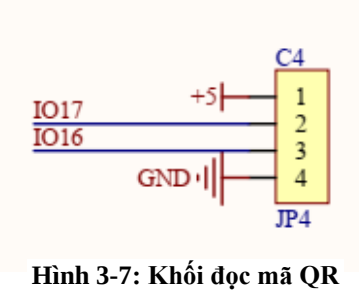
Với yêu cầu đặt ra là hiển thị thông tin một cách đầy đủ và chính xác nhất thì LED 7 đoạn và LED ma trận đều không đáp ứng được bởi chỉ hiển thị số là chính nên ta không thể biết được chính xác tình trạng bãi. Vì thế, LCD sẽ được lựa chọn bởi ưu điểm mà nó mang lại so với các dạng hiển thị khác là khả năng hiển thị kí tự đa dạng, trực quan (chữ, số và kí tự đồ hoạ), dễ dàng đưa vào mạch ứng dụng theo nhiều giao tiếp khác nhau, ít tiêu tốn tài nguyên hệ thống và giá thành phù hợp. Hiện nay, trên thị trường có rất nhiều loại LCD với kích thước và tính năng đa dạng như LCD 16x2, 20x4, 128x64 tuỳ thuộc vào nhu cầu mà mục đích sử dụng. Với đề tài này, LCD 20x4 sẽ lựa chọn vì hệ thống chỉ cần hiển thị tình trạng bãi đậu cho 4 chỗ và thông tin xe ra/vào bãi nên việc lựa chọn trên là hợp lí bởi thông tin sẽ được gọn gàng, đầy đủ và đúng tiêu chí đặt ra. ESP32 sẽ thực hiện giao tiếp với LCD thông qua I2C bởi vì hạn chế được số lượng dây giúp cho thiết kế phần cứng được dễ dàng.

Bảng 3-2: Các chân kết nối LCD với ESP32 thông qua I2C

STT	LCD I2C	ESP32
1	VCC	5V
2	GND	GND
3	SDA	PIN 21
4	SCL	PIN 22

3.7 Khối đọc mã QR

Máy quét được sử dụng trong việc đọc mã QR tạo ra bởi ứng dụng trên điện thoại khi người dùng đặt chỗ ở hệ thống ở cửa ra vào là module quét mã vạch GM65 bởi ưu điểm góc quét rộng, quét chính xác ở nhiều điều kiện ánh sáng khác nhau.



Hình 3-7: Khối đọc mã QR

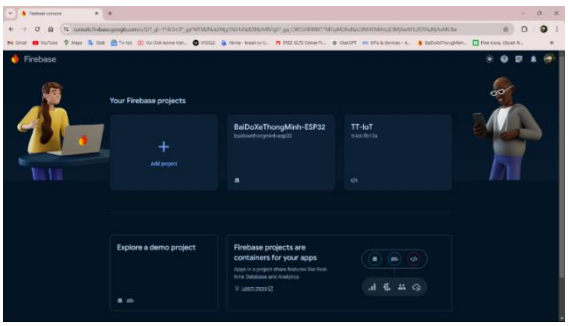
Hiện nay có rất nhiều module quét mã QR khác nhau trên thị trường như ZKTECO QRM10L, Wai 6780,… nhưng GM65 được lựa chọn cho hệ thống bởi vì sự nhỏ gọn, góc quét rộng, giá thành hợp lí, có thể kết nối thông qua giao tiếp UART nên rất phù hợp để kết nối với ESP32.

Bảng 3-3: Các chân kết nối GM65 với ESP32

STT	GM65	ESP32
1	VCC	5V
2	GND	GND
3	TX	PIN 16
4	RX	PIN 17

3.8 Thiết kế phần mềm

3.8.1 Lập trình ESP32 giao tiếp với Firebase trên Arduino



Hình 3-8: Giao diện Firebase

Để thực hiện việc cấu hình kết nối WiFi và Firebase cho ESP32 nhằm mục đích gửi dữ liệu tình trạng bãi xe trực tiếp lên cơ sở dữ liệu ta thực hiện thêm thư viện #include <WiFi.h> và #include <FirebaseESP32.h> vào và thực hiện tạo hai biến SSID và PASSWORD để nhập tên và mật khẩu cho WiFi mà bạn muốn ESP32 sử dụng thông qua hai câu lệnh #define WIFI_SSID “tên WiFi sử dụng” và #define

WIFI_PASSWORD “mật khẩu WiFi sử dụng”. Để cấu hình Firebase ta truy cập vào trang https://firebase.google.com trên trình duyệt web để thực hiện đăng nhập và nhấn vào Add project để khởi tạo một dự án mới. Ta thực hiện đặt tên cho dự án và chọn server Singapore để độ trễ khi gửi và nhận dữ liệu là thấp nhất. Chúng ta đợi trong khoảng một vài giây để Firebase có thể tạo dự án mới.

Mục Realtime Database chính là nơi sẽ thực hiện gửi dữ liệu từ ESP32 lên để ứng dụng thiết kế có thể đọc được dữ liệu về tình trạng bãi đậu xe. Mục Rules trên Realtime Database được mô tả ở Hình 3-9 cần chỉnh sửa dữ liệu đọc và ghi thành true để thực hiện các thao tác ghi và đọc dữ liệu trên cơ sở dữ liệu thời gian thực. Sao chép đường dẫn tới project ta đã tạo ở mục Data để ta gán cho câu lệnh #define FIREBASE_HOST “đường dẫn của project”.

Hình 3-9: Mở tính năng đọc và ghi dữ liệu trên Firebase

Database secrets được lấy bằng cách vào Project Overview chọn Service accounts và chọn tab Database secrects. Phần mật khẩu đó sẽ bị ẩn dưới dạng dấu chấm nên cần phải chọn hiển thị, sao chép và gán nó vào câu lệnh #define FIREBASE_AUTH “mật khẩu đã sao chép”. Chúng ta thực hiện chèn câu lệnh FirebaseData fbdb vào trong code Arduino dùng để quản lý Firebase. Ta thực hiện cấu hình WiFi và Firebase ở phần void setup để có thể biết được tình trạng ESP32 đã có thể kết nối đến WiFi và Firebase hay chưa. Khi tiến hành nạp code cho ESP32 nếu ta thấy trên Serial Monitor xuất hiện địa chỉ IP thì tức là phần kết nối đã được thực hiện thành công.

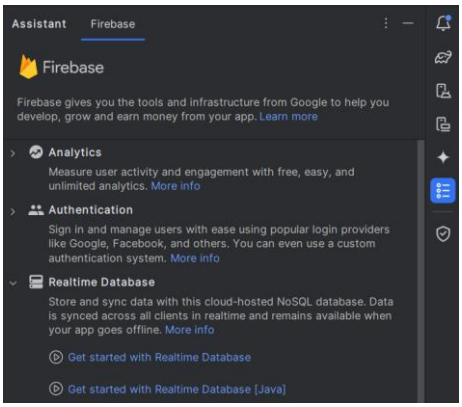
```
void KiemTra()
{
  if(Firebase.getInt(fbdb, "Status/Slot1") == true)
  {

    if (digitalRead(ir_car1) == 0)
    {
      lcd.setCursor(0, 2);
      lcd.print("Slot1: No");
      Firebase.setString(fbdb, "Status/Slot1","No");
    }
    else
    {
      lcd.setCursor(0, 2);
      lcd.print("Slot1: Ok");
      Firebase.setString(fbdb, "Status/Slot1","Ok");
    }
  }
```

Hình 3-10: Gửi dữ liệu tình trạng chỗ đậu thứ 1 lên Firebase

Để gửi dữ liệu vị trí từ ESP32 lên Arduino, ta thực hiện gọi biến quản lý Firebase ra và kiểm tra điều kiện, nếu là true thì thực thi tiếp điều kiện if bên dưới. Điều kiện if ở Hình 3-10 được mô tả nếu cảm biến hồng ngoại trả về giá trị 0, tức là có xe đang đậu ở vị trí số 1 thì sẽ hiển thị ra LCD trạng thái là No, đồng thời sẽ trỏ đến mục "Status/Slot1" trên Firebase và đặt giá trị là "No". Ngược lại, sẽ đặt là "Ok", tương tự với các vị trí số 2, 3, 4.

3.8.2 Thiết kế ứng dụng trên Android Studio

Để bắt đầu tạo một dự án thiết kế ứng dụng trên Android Studio thì ta thực hiện chọn New Project, sau đó thực hiện các bước đặt tên cho ứng dụng, nơi lưu trữ dự án, cài đặt máy ảo, hệ điều hành cho máy ảo và chọn Empty View Activity. Chương trình sẽ tự động tạo một dự án mới trong khoảng một vài phút, khi tạo xong ta sẽ thấy có hai file được tạo ra bao gồm một file Java Class dùng để chứa phần code Java chúng ta dùng để lập trình xử lý chương trình cho phần Layout thiết kế.

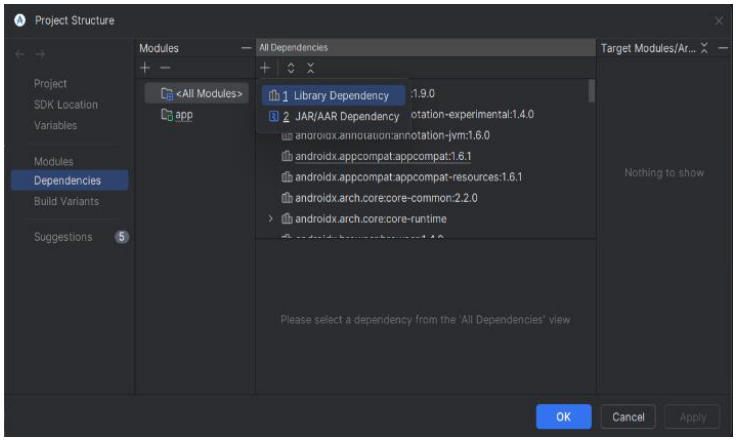


Hình 3-11: Liên kết dự án trên Android Studio với Firebase

Song hành cùng với Java Class sẽ có một file đuôi .xml được tạo ra được gọi là Layout Resource File cho phép chúng ta thiết kế nội dung của ứng dụng bao gồm hình ảnh hiển thị, nút nhấn,… và sẽ là phần xuất hiện trên điện thoại người sử dụng khi truy cập vào ứng dụng.

Để thực hiện được thao tác đọc dữ liệu trên Firebase, ứng dụng đang thiết kế cần phải được kết nối với Firebase bằng cách chọn Firebase ở tab Tools trong Main Menu. Một tab Assistant của Firebase sẽ xuất hiện với vô số tính năng để chúng ta lựa chọn tuỳ vào mục đích sử dụng như mô tả trên Hình 3-11. Với ứng dụng này, ta chỉ cần sử dụng chức năng đọc và ghi dữ liệu trên thời gian thực nên ta sẽ bấm vào chọn Realtime Database. Một Tab trên trình duyệt Web của Firebase sẽ mở ra, ta sẽ thực hiện chọn Project đã tạo sẵn ở mục 3.2.1. Ta có thể bắt đầu sử dụng phần mềm Android Studio một cách bình thường sau khoảng một vài giây đồng bộ với Firebase.

Để có thể kiểm tra xem ứng dụng đã được liên kết với Firebase hay chưa, tiến hành chọn Project ở góc trái màn hình, mở rộng mục Gradle Scripts ra và nhấn vào build.gradle.kts (Module :app) để mở tab. Ở phần dependencies, nếu có xuất hiện dòng implementation(libs.firebase.database) tức là đã thực hiện liên kết với Firebase thành công.



Hình 3-12: Mở thư viện trong Project Structure

Thiết kế phần đăng nhập cho người dùng bằng cách thêm thư viện vào implementation(libs.firebase.auth) vào build.gradle.kts (Module :app) để chứa thông tin tài khoản mà người dùng đã đăng nhập trên Firebase. Để vào mục Project

Structure, ta gõ Ctrl+Alt+Shift+S và chọn Dependencies để hiển thị các thư viện hiện có trong modules. Thư viện được thêm vào bằng cách chọn Library Dependency khi thực hiện bấm Add ở bên dưới mục All Dependencies. Một tab Add Library Dependency sẽ mở ra cho phép chúng ta thêm thư viện cần sử dụng vào, ta thực hiện gõ com.google.firebase:firebase-auth ở Step 1 và bấm Search. Thư viện Auth của Firebase sẽ xuất hiện, ta nhấn chọn phiên bản cần cài đặt và nhấn Apply để tiến hành cài đặt. Sau khi đợi khoảng một vài giây để hệ thống tiến hành thêm thư viện và đồng bộ, chúng ta sẽ thấy xuất hiện dòng implementation(libs.firebase.auth) trên tab build.gradle.kts (Module :app) để cho biết quá trình thêm thư viện đã thành công.

Thiết kế layout phần đăng nhập và đăng ký cho người dùng được thực hiện bằng việc tạo 2 file gồm activity_login.xml sẽ xuất hiện khi người dùng thực hiện đăng nhập vào hệ thống và activity_register.xml được sử dụng khi người dùng muốn đăng ký tài khoản để sử dụng. Ta sẽ thực hiện tạo 4 thanh TextView với 2 thanh dùng để hiển thị Email và Password do chúng ta nhập ở phần code của TextView đó với câu lệnh android:text=”Email” và android:text=”Password”. Người dùng có thể nhập thông tin của mình vào trên 2 TextView còn lại được chừa trống. Để thêm gợi ý cho người dùng về việc nhập thông tin, ta sử dụng lệnh android:hint=”Enter Your Email” và android:hint=”Enter Your Password” với file activity_login.xml.

Một nút Login được thêm vào để người dùng sau khi điền tất cả thông tin thì sẽ tiến hành chuyển sang một layout mới và một nút Register để người dùng có thể đăng kí thông tin tài khoản bao gồm Email và Password lên Firebase. Người dùng khi đăng nhập thất bại do chưa có tài khoản trên hệ thống quản lý sẽ có 1 TextView đi kèm hiển thị trên giao diện hướng dẫn người dùng bấm nút đăng kí với nội dung “Don't have an account? Register now!”. Một file Java Class được tạo ra mang tên LoginActivity.java sẽ dùng để chứa code java thiết lập điều khiển khi người dùng nhấn vào Button Login và Button Register. Chức năng Auth của Firebase được khai báo sử dụng bằng câu lệnh private FirebaseAuth mAuth, private Button Loginbtn, registerBtn cho 2 nút nhất và private EditText emailEdt, passEdt cho 2 TextView được chừa trống cho người dùng điền thông tin đăng nhập. Câu lệnh mAuth =

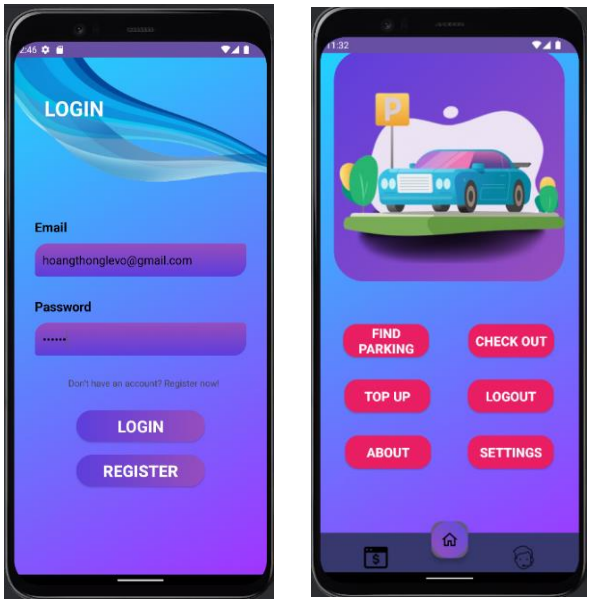
FirebaseAuth.getInstance() thực hiện các tác vụ xác thực như đăng nhập, đăng ký, đăng xuất và quản lý thông tin người dùng.

Hình 3-13: Giao diện đăng nhập trên ứng dụng

Dòng mã emailEdt=findViewById(R.id.emailtext) được sử dụng để tìm kiếm TextView trong layout có ID emailtext được chừa trống để người dùng nhập email và tương tự đối với ID passtext cho người dùng nhập mật khẩu, dòng mã này cũng tìm kiếm ID 2 nút nhấn bao gồm Loginbtn và registerBtn đại diện cho nút Login và nút Register khi được người dùng nhấn. Khi nút Login được nhấn, hai biến emailEdt và passEdt được đặt kiểu dữ liệu chuỗi khi người dùng nhập.

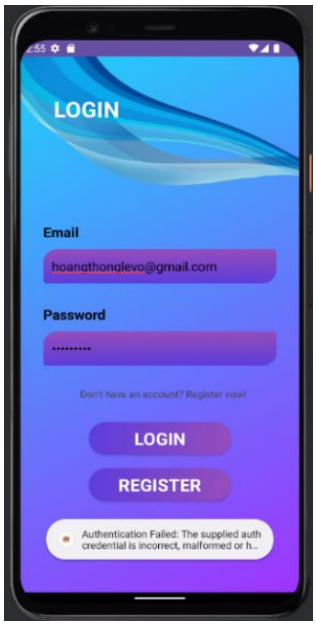
Hình 3-14: Xuất hiện thông báo khi người dùng không nhập thông tin

Câu lệnh if sẽ được thiết lập nếu người dùng bỏ trống không nhập thông tin ở hai biến emailEdt và passEdt thì ngay lập tức hệ thống sẽ thông báo "Please fill the login form" và không cho phép người dùng chuyển sang layout tiếp theo như trong Hình 3-14.



Hình 3-15: Chuyển sang layout mới khi người dùng đăng nhập vào hệ thống

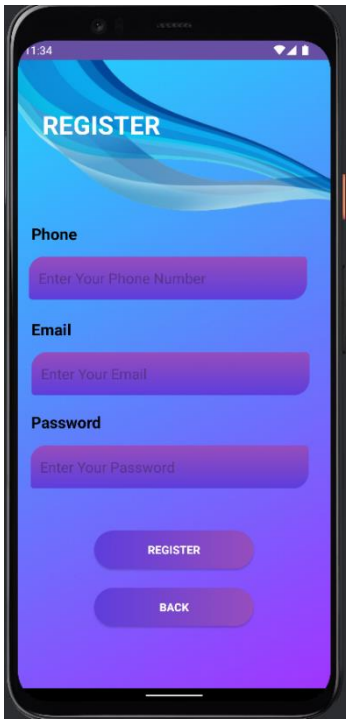
Người dùng đã nhập thông tin và khi đối chiếu thông tin trên hệ thống quản lý thành công thì hệ thống sẽ lập tức chuyển sang một layout mới ở đây tức là activity_main.xml thuộc lớp MainActivity.java như Hình 3-15.



Hình 3-16: Trường hợp người dùng chưa đăng kí thông tin trên hệ thống bấm đăng nhập

Trường hợp người dùng đã nhập đầy đủ thông tin nhưng sau khi đối chiếu trên hệ thống quản lý không có tài khoản trên thì sẽ trả về thông báo "Authentication Failed:" kèm với lý do xác thực người dùng thất bại đồng thời không cho người dùng chuyển qua layout mới như miêu tả ở Hình 3-16.

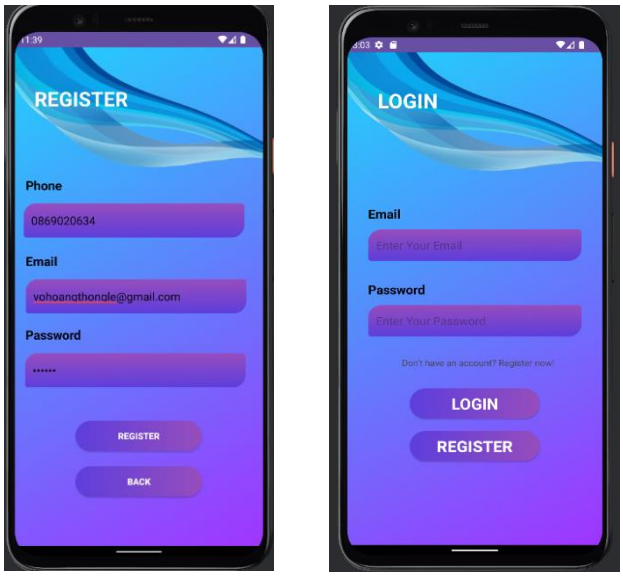
Nút Register sẽ cho phép người dùng chuyển qua một layout mới gọi là activity_register.xml thuộc lớp RegisterActivity.java có giao diện như Hình 3-17. Layout activity_register.xml cũng được tạo tương tự như ở activity_login.xml chỉ thay đổi ở phần tên nút nhấn thành Register và thêm phần nhập số điện thoại để sau khi người dùng nhập thông tin thì hệ thống sẽ lưu thông tin và lập tức quay về layout activity_login.xml để người dùng nhập thông tin và đăng nhập. Nút Back sẽ được tạo ra khi người dùng muốn quay về trang đăng nhập.



Hình 3-17: Giao diện đăng kí

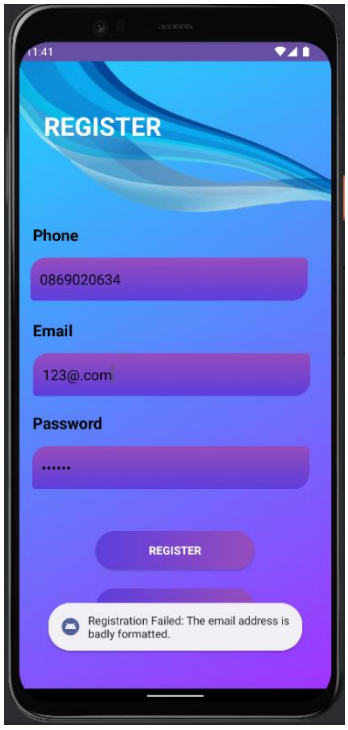
Lớp RegisterActivity.java được thay đổi ở câu lệnh điều kiện khi người dùng không điền thông tin thì hệ thống sẽ lập tức thông báo hãy điền đầy đủ thông tin tương tự như phần đăng nhập. Khi các thông tin được điền đầy đủ theo đúng với yêu cầu hệ thống, hệ thống sẽ thông báo "Registration Successful" và lập tức quay trở về trang

đăng nhập để người dùng nhập thông tin đã đăng ký trên hệ thống, tức là layout activity_login.xml như ở Hình 3-18.



Hình 3-18: Khi đăng kí thành công sẽ quay về trang Login

Người dùng khi nhập thông tin không đúng với yêu cầu của hệ thống thì ngay lập tức hệ thống sẽ báo lỗi "Registration Failed:" cùng với lỗi gặp phải cho người dùng được biết như trong Hình 3-19.



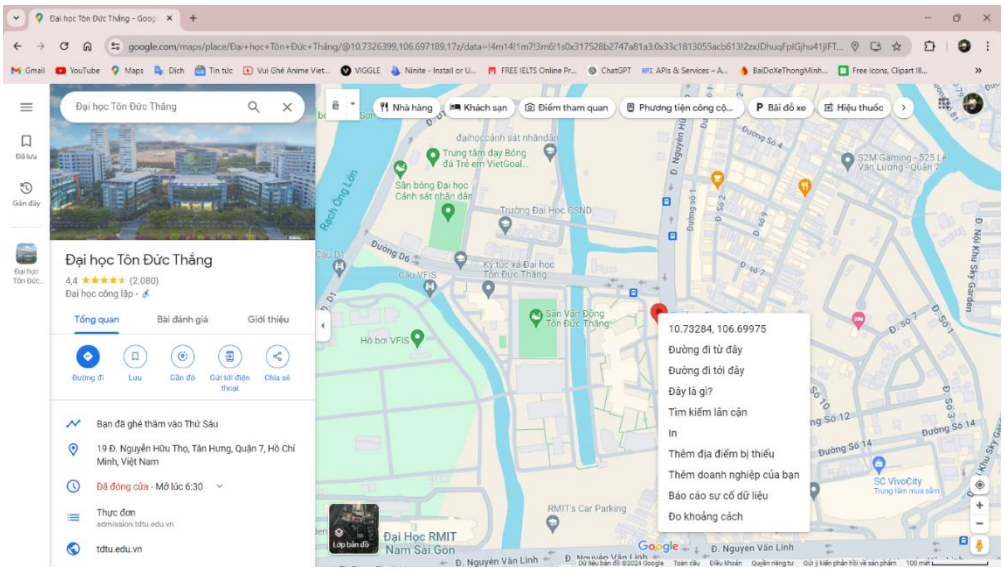
Hình 3-19: Đăng kí thông tin sử dụng kí tự không hợp lệ

Sau khi đăng nhập thành công, người dùng sẽ được chuyển sang layout mới với tên activity_main.xml. Một nút nhấn được tạo ra đối với layout này để chuyển người dùng tới layout để đặt chỗ. Người dùng được chuyển sang layout mới với tên activity_maps.xml khi nút Find Parking trên layout activity_main.xml được nhấn. Một layout chứa bản đồ được cung cấp bởi Google sẽ được tạo ra. Để có thể sử dụng tính năng hiển thị bản đồ với các vị trí bãi đậu xe, chúng ta cần phải thêm thư viện implementation(libs.play.services.maps) vào trên tab build.gradle.kts (Module :app) với các bước thực hiện tương tự như trên. Chúng ta vào trình duyệt Web và tìm đến trang https://console.cloud.google.com để có thể lấy API key và nhập vào tập tin AndroidManifest.xml cung cấp quyền truy cập đến bản đồ được cung cấp bởi Google một cách miễn phí.

Để mở tất cả tính năng của bản đồ sau khi đăng nhập trên Google Cloud console, ta khởi tạo dự án mới và bật các tính năng trên Marketplace bao gồm Google Maps Booking API, Google Maps Booking API(Dev), Google Maps for Fleet Routing, Cloud Channel API, Automotive Maps API, Google Picker API, Street View Publish API. Key của API được tạo có thể sao chép bằng cách bấm vào nút show key, mở tệp AndroidManifest.xml ở mục project và thực hiện dán vào câu lệnh android:value ở meta-data. Một vài câu lệnh được thực thi để cấp quyền truy cập cho người dùng khi thực hiện vào bản đồ bao gồm cho phép truy cập vào vị trí tốt, internet, trạng thái mạng truy cập, truy cập vào vị trí thô.

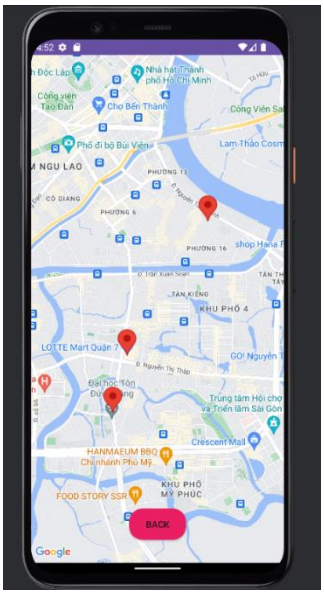
Khi đã hoàn thành các bước cài đặt trên, chúng ta trở lại file activity_maps.xml và tiến hành thiết kế một fragment bao phủ layout. Khi người dùng chuyển sang layout này lập tức một bản đồ sẽ hiển thị. Nút nhấn bao gồm Check Slots sẽ được thiết lập khi người dùng ấn vào Marker với thông tin vị trí được tạo sẵn được giả lập là vị trí một bãi đậu xe thông minh. Một Marker hiển thị trên bản đồ sẽ chứa thông tin vĩ độ và kinh độ (gọi là latitude và longitude). Để có thể lấy thông tin vĩ độ và kinh độ, ta thực hiện vào Google Maps trên trình duyệt Web, nhấn chuột phải vào vị trí Marker cần lấy thông tin thì ngay lập tức vĩ độ và kinh độ sẽ hiển thị. Ví dụ ta nhấn chuột

phải vào Marker của trường Đại học Tôn Đức Thắng thì ngay lập tức ta sẽ nhận được thông tin vĩ độ và kinh độ của trường như trong Hình 3-20.



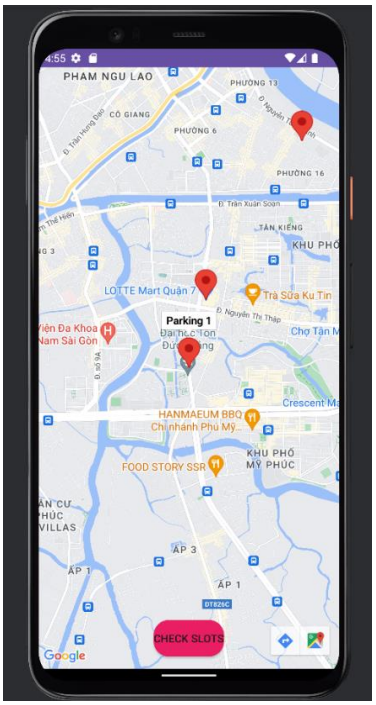
Hình 3-20: Lấy toạ độ trên Google Maps

Thư viện private GoogleMap mMap được thêm vào vào lớp MapsActivity.java khi ta muốn sử dụng các giao diện có bản đồ. Thông tin vĩ độ và kinh độ sau khi được sao chép sẽ được nhập vào lớp MapsActivity.java với dòng lệnh LatLng parking1 = new LatLng(10.733609684695706, 106.69989264603498), thực hiện tương tự với các vị trí khác nếu muốn tạo thêm. Đối với ứng dụng này, 3 Marker được khởi tạo tương ứng với 3 vị trí của bãi đỗ xe thông minh như trong Hình 3-21.



Hình 3-21: Các Marker sẽ hiển thị trên bản đồ sau khi được gán toạ độ

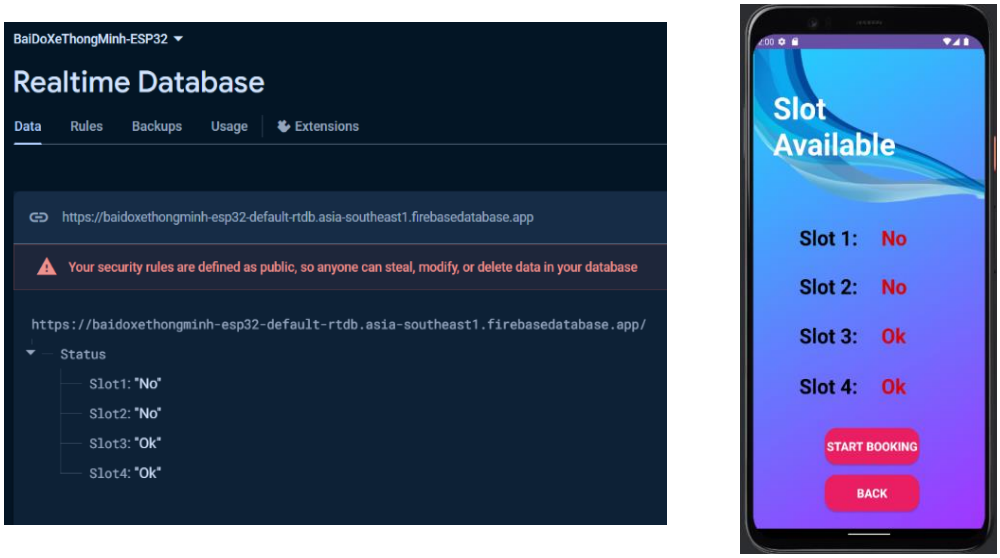
Một bản đồ với 3 Marker được tạo với tên Parking 1, Parking 2, Parking 3 mô phỏng 3 vị trí bãi đậu xe thông minh được hiển thị khi người dùng truy cập như miêu tả ở Hình 3-22. Khi nhấn vào Marker, nút Check Slots được hiển thị cho phép người dùng chuyển sang layout tiếp theo, tức là layout activity_detail.xml để kiểm tra tình trạng bãi đậu xe xem còn chỗ hay không. Khi nhấn vào vùng khác trên bản đồ thì nút Check Slots sẽ biến mất mà thay vào đó là nút Back sẽ được hiển thị cho phép người dùng quay trở về layout activity_main.xml.



Hình 3-22: Mỗi Marker tương ứng với 1 vị trí bãi đậu xe

Người dùng sẽ được chuyển sang activity_detail.xml khi nhấn vào nút Check Slots. Thông tin về tình trạng chỗ lấy dữ liệu trực tiếp từ Firebase do ESP32 gửi lên sẽ được hiển thị. Vị trí của 4 chỗ đậu sẽ được thiết kế hiển thị trên 4 TextView và 2 nút nhấn được bố trí bao gồm 1 nút Start Booking để chuyển đến layout tiếp theo và nút Back để trở về layout trước, tức là activity_maps.xml. Ở layout này, ta sẽ thực hiện lấy dữ liệu trực tiếp từ Firebase đã được ESP32 gửi lên Firebase thông qua dòng lệnh FirebaseDatabase database = FirebaseDatabase.getInstance() và trỏ tới mục “Status/Slot1” để lấy dữ liệu thông qua dòng lệnh DatabaseReference myRef = database.getReference("Status/Slot1"). Sau cùng là thiết lập hiển thị lên TextView dữ

liệu kiểu chuỗi thông qua dòng lệnh String value = snapshot.getValue(String.class) và sẽ báo lỗi nếu như không lấy được dữ liệu từ Firebase. Việc lấy dữ liệu từ Firebase sẽ tương tự như thế với 3 vị trí còn lại. Dữ liệu hiển thị sau khi chạy ứng dụng sẽ thông báo tình trạng bãi đậu xe bao gồm trạng thái "No" nếu cảm biến hồng ngoại phát hiện được có xe đang đậu tại vị trí đó và trạng thái "Ok" nếu cảm biến hồng ngoại không phát hiện xe như Hình 3-23.



Hình 3-23: Giao diện hiển thị tình trạng chỗ của bãi đậu xe

Dữ liệu sẽ được cập nhật liên tục để gửi lên cho người dùng, giúp người dùng biết chính xác tình trạng bãi đậu xe để có thể chuyển sang layout đặt chỗ activity_booking.xml. Ở giao diện đặt chỗ, ta sẽ thiết kế bao gồm 1 Spiner để cho phép người dùng chọn chỗ dựa trên việc lấy dữ liệu từ Firebase, 2 Button để chọn ngày, giờ vào và 1 TextView để hiển thị giá tiền đặt cọc chỗ với mức giá là 10000 đồng. Tương tự như những layout trước là 1 Button sử dụng cho việc chuyển sang layout tiếp theo và 1 Button trở về layout trước đó.

Người dùng khi nhấn vào Button với id btnSelectDate đoạn mã này sẽ thu thập dữ liệu số nguyên và lưu vào các biến "year", "month", "date". Câu lệnh selectedDate.set(year1, monthOfYear, dayOfMonth) sẽ cho phép đặt ngày được chọn vào "selectedDate". Dữ liệu ngày tháng năm sẽ được định dạng "dd-MM-yyyy" khi hiển thị trên màn hình giao diện người dùng sau đó đặt văn bản của "btnSelectDate" thành ngày đã chọn dưới dạng chuỗi thông qua việc sử dụng câu lệnh

btnSelectDate.setText(sdf.format(selectedDate.getTime())). Việc thiết lập Button cho việc lựa chọn giờ vào và giờ kết thúc cũng được thực hiện tương tự như trên và được định dạng hiển thị dưới dạng “HH:mm”. Hàm SlotsFromFirebase đầu tiên sẽ thực hiện tham chiếu đến node “Status” từ Firebase và sẽ thực hiện theo dõi các sự thay đổi trong node “Status”. Câu lệnh List<String> availableSlots = new ArrayList<>() sẽ thực hiện tạo một danh sách để lưu các slots có sẵn và sau đó sẽ thực hiện lấy từng giá trị slotId nằm trong node “Status”. Câu lệnh điều kiện sẽ được thực thi khi trạng thái hiển thị ở Status là "Ok" thì sẽ thực hiện thêm slotId vào danh sách availableSlots. Cuối cùng sẽ thực hiện tạo Adapter cho Spiner, nếu như người dùng thực hiện chọn Slot trên Spiner thì sẽ hiển thị những Slot ở trạng thái “Ok”, tức là trạng thái không có xe đậu.

Người dùng khi đã thực hiện chọn các thông tin bao gồm Slot, ngày đặt chỗ, giờ vào thì một nút nhấn btnReserveAndPay được khởi tạo với điều kiện nếu tiền đặt cọc lớn hơn hoặc bằng 10000 đồng trong node users/balance thì sẽ tiến hành lưu lại thông tin đặt chỗ và sẽ thực hiện chuyển sang Activity tiếp theo.

Hình 3-24: Dữ liệu balance sẽ trừ tiền khi người dùng đặt chỗ thành công

Thông tin đặt chỗ được hiển thị để thông báo cho người dùng và giá tiền đặt cọc cần phải trả khi đặt chỗ thì người dùng sẽ thực hiện thanh toán đặt cọc chỗ và lấy mã QR để quét vào bãi đậu xe. Giá tiền đặt cọc chỗ được thu trước sẽ giúp cho người dùng hạn chế việc đặt chỗ liên tiếp và tránh tình trạng bỏ chỗ đã đặt gây ảnh hưởng đến người dùng khác. Để tránh có sự trùng lặp về thời gian và vị trí đậu giữa những người

đặt chỗ thì hàm checkForDuplicateBooking được tạo ra để kiểm tra và vô hiệu hoá nút Booking đối với những người đặt chỗ sau chọn trùng thời gian đặt chỗ với người đặt chỗ trước.

Hình 3-25: Giao diện đặt chỗ

Sau khi người dùng trước đặt chỗ, một node “Bookings” chứa dữ liệu Slot, ngày đặt chỗ, giờ vào được lưu trên Firebase. Hàm checkForDuplicateBooking sẽ thực hiện tham chiếu để kiểm tra Slot, ngày đặt chỗ, giờ vào của người đặt chỗ trước để kiểm tra với dữ liệu của người đặt chỗ sau. Biến boolean isDuplicate = false sẽ được sử dụng để theo dõi liệu có bản ghi đặt chỗ nào bị trùng lặp hay không trong quá trình duyệt qua các dữ liệu. Một câu lệnh if được thực thi để kiểm tra xem đối tượng bookingInfo có khác null không (để tránh lỗi NullPointerException), sau đó so sánh các giá trị của slot, date và startTime trong bookingInfo với các giá trị được truyền vào là slot, selectedDateStr và selectedStartTimeStr. Nếu tất cả các giá trị này đều trùng khớp, điều đó có nghĩa là đã tồn tại một bản ghi đặt chỗ với cùng vị trí, ngày và giờ bắt đầu. Nếu điều kiện if ở trên là đúng (nghĩa là tìm thấy bản ghi trùng lặp), biến isDuplicate sẽ được gán giá trị true và câu lệnh break này ngay lập tức thoát khỏi

vòng lặp vì khi đã tìm thấy một bản ghi trùng lặp, không cần phải tiếp tục kiểm tra các bản ghi còn lại.

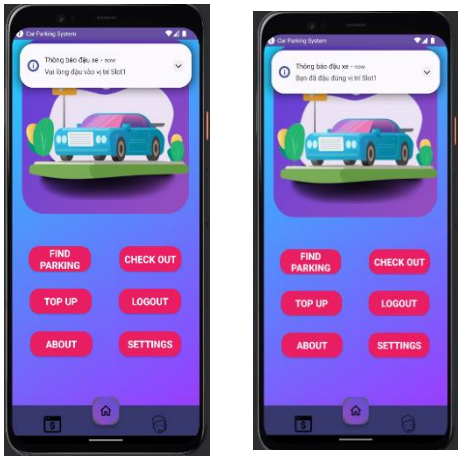
Hình 3-26: Giao diện hiển thị cảnh báo khi đặt chỗ trùng lặp

Trình tự thực hiện tham chiếu sẽ được thực hiện nếu vị trí đậu của người đặt chỗ sau trùng với người đặt chỗ trước thì tiến hành kiểm tra giờ vào và giờ ra, nếu như trùng thời gian đặt chỗ thì ngay lặp tức biến isOverlap từ giá trị false sẽ trở thành true và ngay lặp tức thoát khỏi vòng lặp và ngay lặp tức vô hiệu hoá không cho phép người dùng nhấn vào nút đặt chỗ và hiển thị ra thông báo “Thời gian này đã có người đặt chỗ. Vui lòng chọn thời gian khác.”.

Người dùng sẽ được chuyển sang layout activity_info.xml khi đã thực hiện điền đầy đủ các thông tin đặt chỗ và nhấn vào nút Booking. Một mã QR sẽ hiển thị chứa tất cả thông tin về vị trí đậu xe, ngày đặt chỗ, thời gian bắt đầu, thời gian kết thúc và giá tiền để người dùng có thể sử dụng như một tấm vé vào cổng của bãi đậu xe. Bên dưới sẽ là các TextView được thiết lập để hiển thị các thông tin chứa trong mã QR của người dùng. Để có thể tạo được mã QR trong ứng dụng dựa trên các thông tin có sẵn, ta thực hiện thêm thư viện tên dependencies trên Module: app với dòng implementation(libs.zxing.android.embedded).

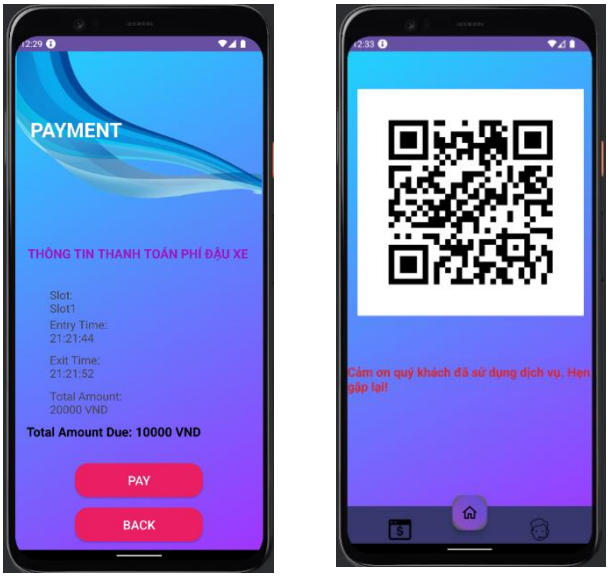
Hình 3-28: Giao diện hiển thị mã QR khi người dùng đặt chỗ thành công

Câu lệnh Intent intent = getIntent() sẽ cho phép ta lấy Intent đã khởi động Activity này, sau đó sẽ thực hiện lấy chuỗi thông tin chi tiết về đặt chỗ từ Intent bằng cách sử dụng key "reservationDetails" mà trước đó đã được truyền ở nút Booking của layout trước đó. Cấu trúc try-catch được sử dụng để bắt và xử lý các ngoại lệ có thể xảy ra trong quá trình tạo mã QR. Một mã QR có kích thước 400x400 pixels sẽ được mã hoá bằng chuỗi dữ liệu đã thu thập từ quá trình đặt chỗ ở layout trước. Người dùng sẽ nhận được thông báo đậu xe theo đúng chỗ mình đã đặt sau khi đã quét mã QR vào cổng. Người dùng đặt chỗ ở vị trí số 1 khi quét mã QR vào cổng, dữ liệu được gửi lên Firebase là có người đang vào cổng ở vị trí số 1 và sẽ thông báo lên màn hình cho người dùng được biết. Người dùng di chuyển xe vào đúng vị trí thì sẽ nhận được thông báo “Bạn đã đậu đúng vị trí Slot1” từ hệ thống.



Hình 3-27: Thông báo hiển thị khi người dùng đậu đúng hoặc sai vị trí

Người dùng khi ra khỏi vị trí số 1 và muốn ra khỏi bãi giữ xe thì dữ liệu về Slot1 sẽ được cập nhật lại không có xe đậu và đồng thời gửi thời gian vào, thời gian kết thúc đậu xe và giá tiền lên cơ sở dữ liệu và được ứng dụng lấy về để hiển thị cho người dùng thực hiện thanh toán. Giá tiền đặt cọc chỗ cũng sẽ được hoàn trả khi người dùng thực hiện thanh toán phí đậu xe, với phí đậu xe dưới 2 tiếng được quy định là 20000 đồng thì khi người dùng được hoàn trả tiền cọc thì chỉ phải thanh toán 10000 đồng cho ứng dụng. Với thời gian đậu xe từ 2 tiếng đến dưới 3 tiếng thì mức tiền chi trả là 30000 đồng và từ 3 tiếng trở đi người dùng sẽ chỉ phải chi trả 50000 đồng. Khi thực hiện thanh toán xong, người dùng sẽ nhận được mã QR để ra khỏi bãi đậu xe.



Hình 3-29: Người dùng thực hiện thanh toán và nhận mã QR để ra khỏi bãi đậu xe

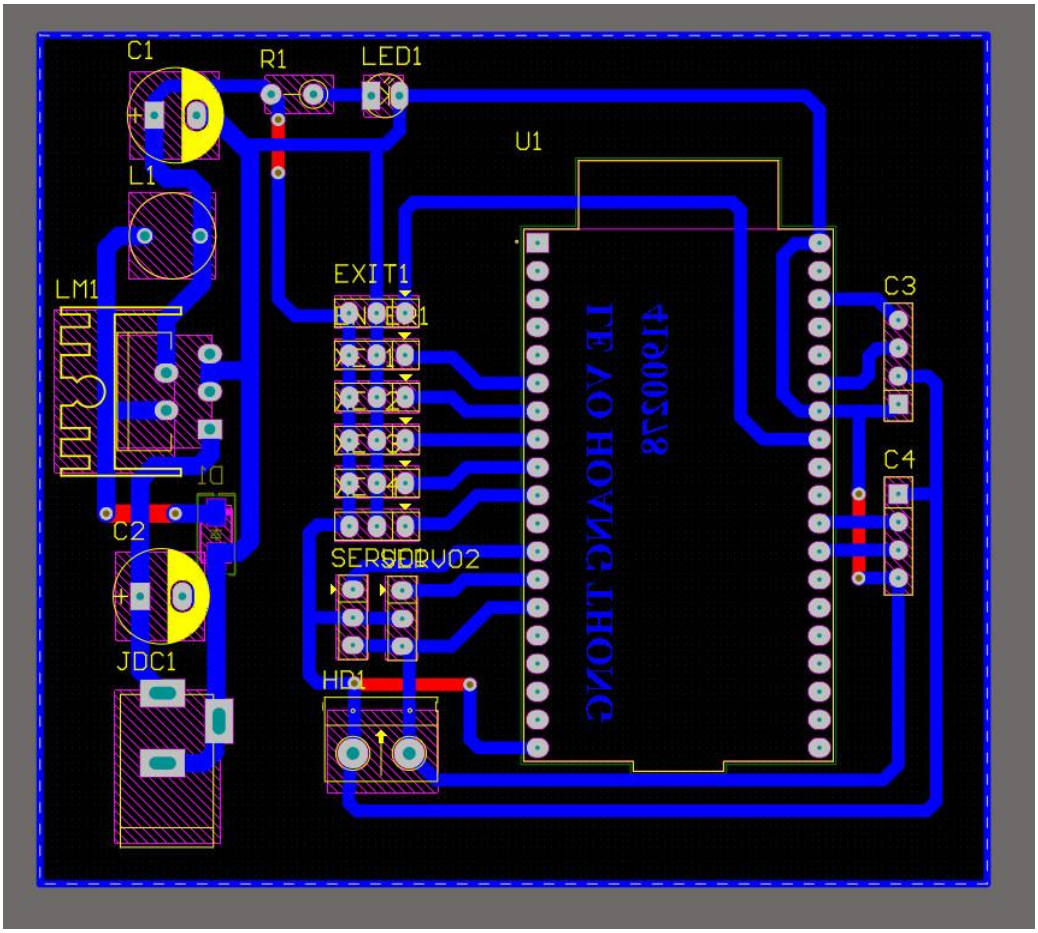
CHƯƠNG 4. THI CÔNG VÀ THỬ NGHIỆM

Phần cứng và ứng dụng sau khi thiết kế sẽ được thử nghiệm và tiến hành đánh giá để có những điều chỉnh tối ưu cho hệ thống.

4.1 Mô hình thực tế

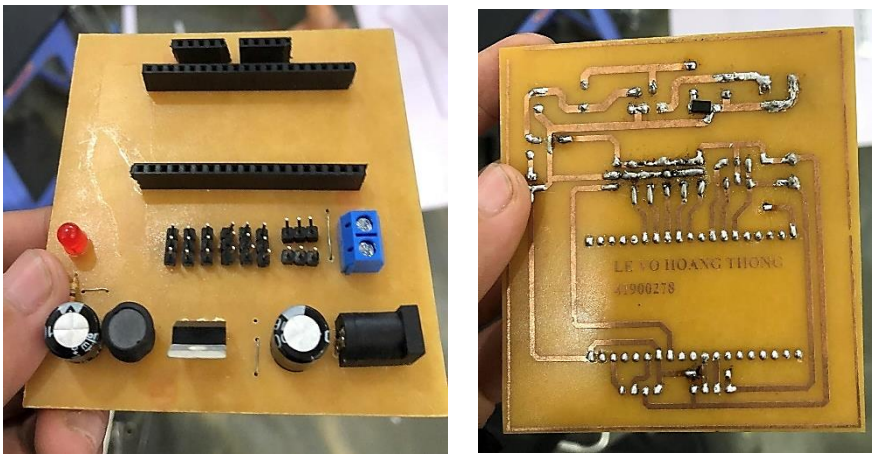
4.1.1 Phần cứng

Sau khi đã thiết kế sơ đồ khối và sơ đồ nguyên lý, mạch in sẽ được vẽ dựa trên sơ đồ nguyên lý đã thiết kế.



Hình 4-1: Bản vẽ cho mạch in

Mạch đồng được thực hiện dựa trên sơ đồ mạch in đã thiết kế phía trên.



Hình 4-2: Mạch đồng sau khi hoàn thiện

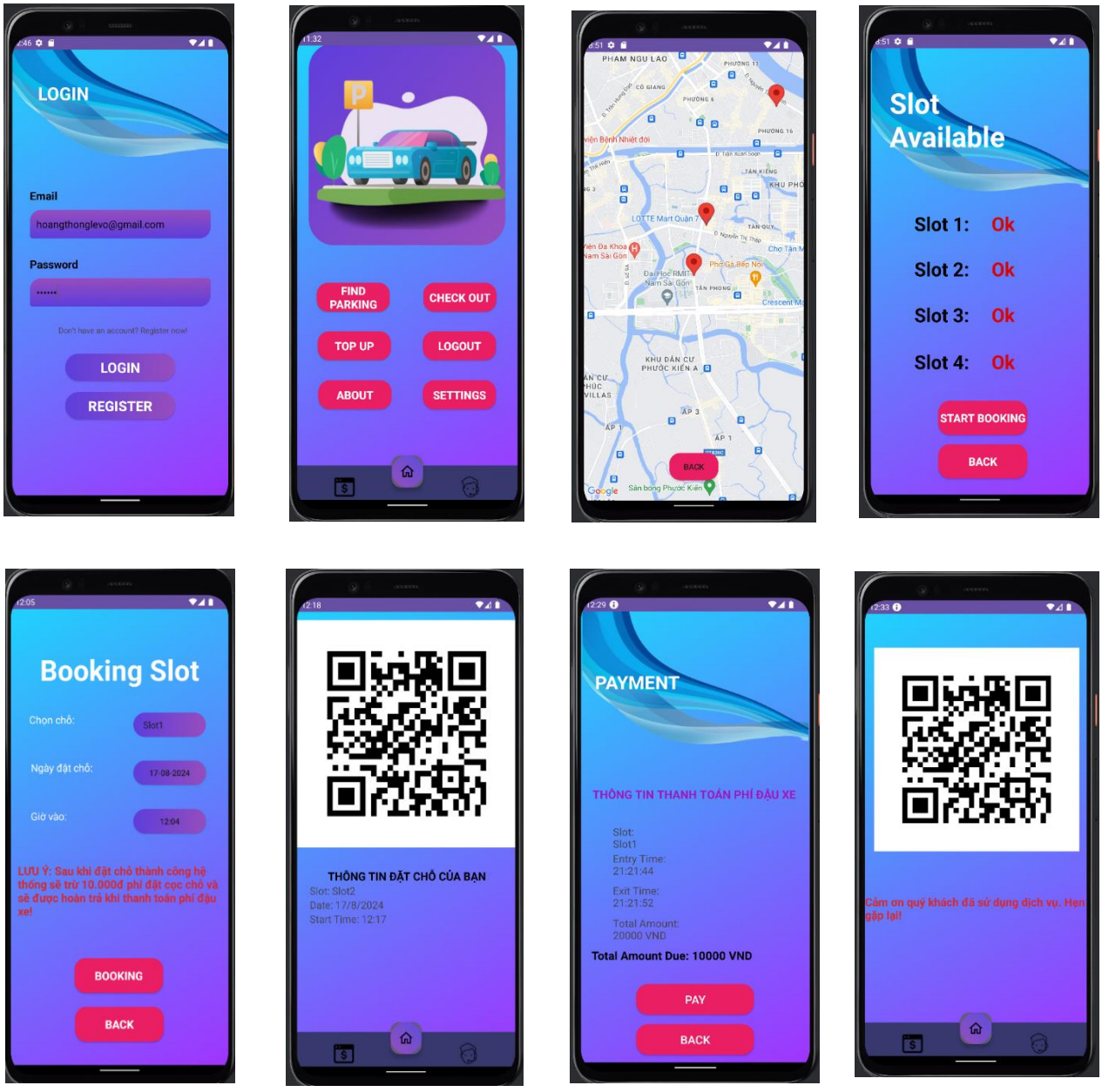
Mô hình bãi đậu xe thực tế được thi công bao gồm 2 barrier được mô phỏng bởi 2 servo và 2 cảm biến hồng ngoại được phân chia đặt ở lối vào/ra của hệ thống và 4 cảm biến hồng ngoại còn lại được sử dụng trong việc xác định vị trí xe đỗ trong bãi.

Hình 4-3: Mô hình bãi đậu xe thông minh hoàn thiện

4.1.2 Phần mềm

Theo yêu cầu và phương án thiết kế đã đề ra, một ứng dụng di động được phát triển sử dụng ngôn ngữ lập trình Java trên Android Studio với các chức năng hiển thị bản đồ với ba vị trí bãi đậu xe thông minh giúp người dùng dễ dàng tìm kiếm bãi đậu xe gần nhất, cập nhật tình trạng bãi đậu xe và thời gian vào ra của xe để tính tiền theo thời gian thực bằng việc cho ứng dụng trích xuất dữ liệu từ cơ sở dữ liệu thời gian

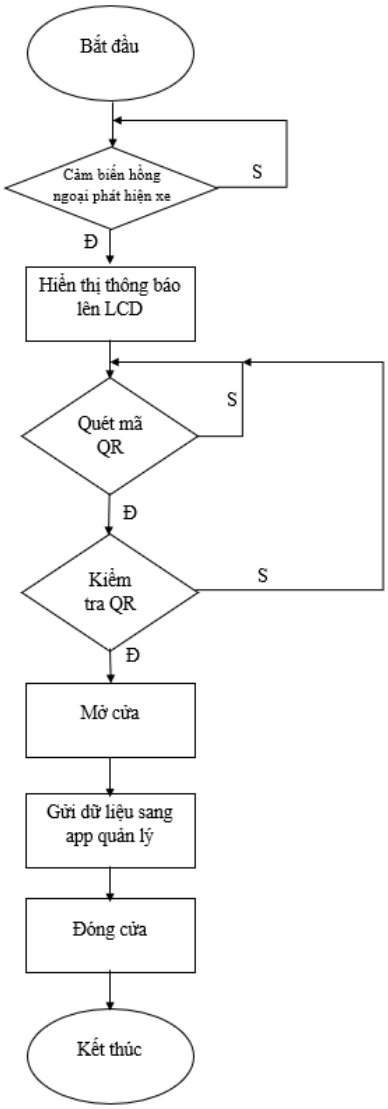
thực Firebase để người dùng có thể theo dõi nhanh chóng và chính xác nhất, tích hợp thêm tính năng đặt chỗ, cho phép người dùng đặt chỗ trước và tạo mã QR để người dùng sử dụng khi ra vào hệ thống.



Hình 4-4: Toàn bộ giao diện của ứng dụng

Người dùng sẽ tiến hành đăng nhập và thực hiện đặt chỗ trên ứng dụng đã thiết kế. Khi quá trình đặt chỗ thành công, ứng dụng sẽ tạo ra một mã QR chứa các thông tin đặt chỗ bao gồm vị trí chỗ đậu, ngày đặt chỗ và thời gian bắt đầu đậu xe. Sau khi có

được mã QR, xe sẽ tiến vào bãi đậu xe. Cảm biến hồng ngoại nhận diện có xe, xuất ra màn hình LCD yêu cầu quét mã QR để vào cổng. Khi hệ thống đã kiểm tra bãi đậu xe còn chỗ và quá trình quét mã QR thành công thì sẽ thực hiện mở servo để xe vào vị trí đã đặt. Nếu như việc quét mã QR thất bại, hệ thống sẽ yêu cầu quét lại mã QR cho đến khi thành công. Sau khi xe đã đậu vào vị trí đã đặt, hệ thống sẽ cập nhật dữ liệu lên cơ sở dữ liệu và hiển thị thông tin trạng thái chỗ đậu xe lên LCD. Quá trình xe ra khỏi bãi đậu xe, người dùng thực hiện thanh toán trên ứng dụng và sẽ nhận được mã QR để ra cổng, việc thực hiện quét mã QR cũng được thực hiện tương tự như khi xe vào bãi đậu.



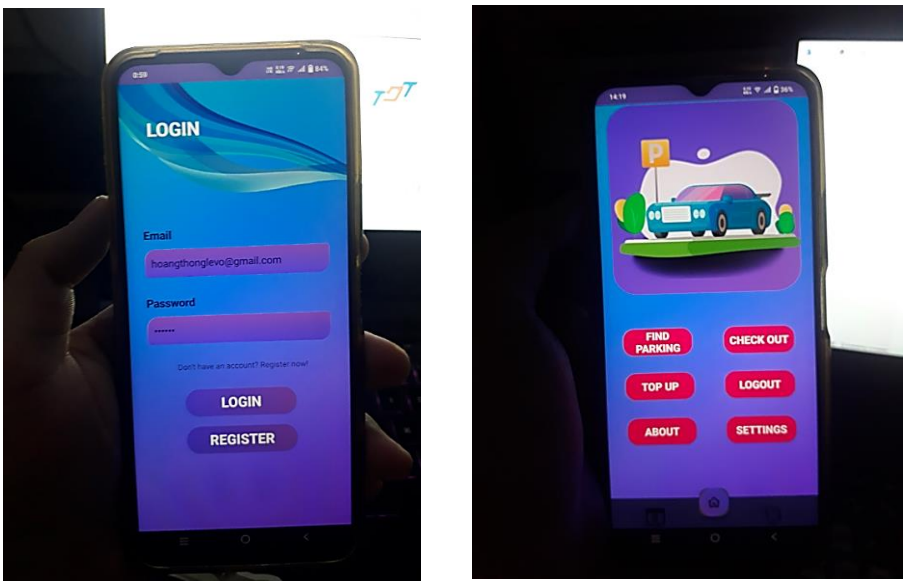
Hình 4-5: Lưu đồ giải thuật

4.1.3 Kết quả

Mô hình bãi đậu xe được khởi động, tất cả các cảm biến hoạt động bình thường, các chỗ đậu trống được hiển thị trên màn hình LCD là “Ok” đúng với yêu cầu.

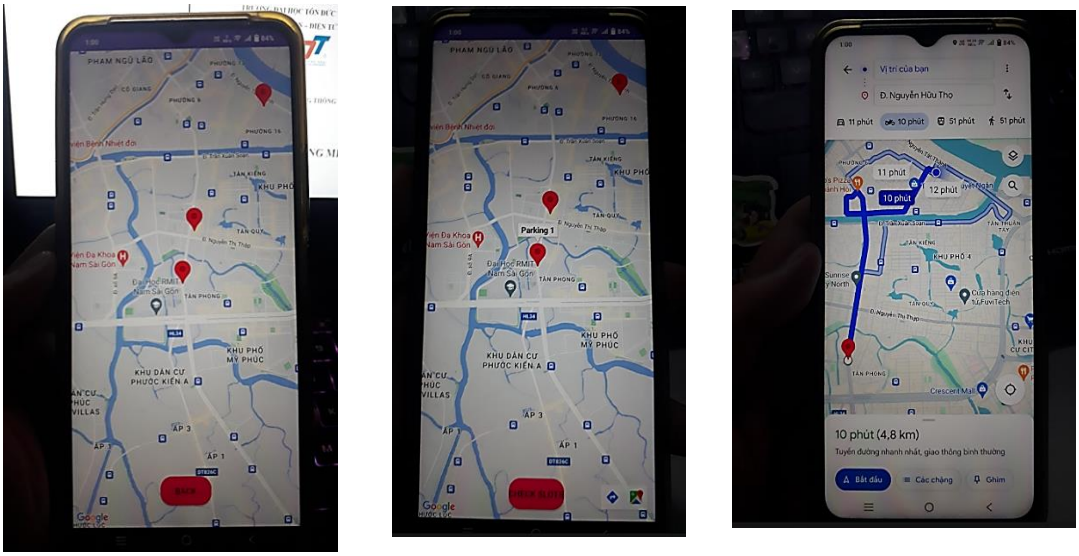
Hình 4-6: Mô hình hệ thống khi khởi động

Một quy trình vào bãi đậu xe sẽ được thực hiện bằng việc đăng nhập trên ứng dụng đã thiết kế. Sau khi đăng nhập thành công, người dùng sẽ được chuyển đến trang chính của ứng dụng. Ta chọn nút Find Parking trên trang chính của ứng dụng để bắt đầu đặt chỗ.



Hình 4-7: Đăng nhập ứng dụng

Một bản đồ được cung cấp các tính năng bởi Google Maps sẽ được hiển thị ra giúp người dùng xem được vị trí của các bãi đậu xe thông qua các Marker.

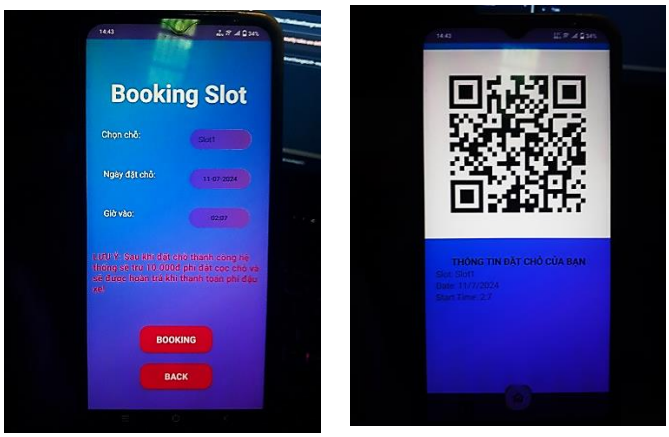


Hình 4-8: Xem giao diện bản đồ tìm vị trí bãi xe

Bên góc phải màn hình sẽ xuất hiện mũi tên khi chọn một Marker tương ứng với bãi đậu, người dùng khi nhấn vào đó sẽ ngay lập tức được chuyển sang giao diện Google Maps để hướng dẫn đường đi đến bãi đậu xe đó. Ta chuyển sang giao diện đặt chỗ để xem tình trạng chỗ ở bãi đậu xe đó. Các cập nhật trên ứng dụng hoàn toàn khớp với dữ liệu trên Firebase và màn hình LCD hiển thị tại bãi đậu xe.

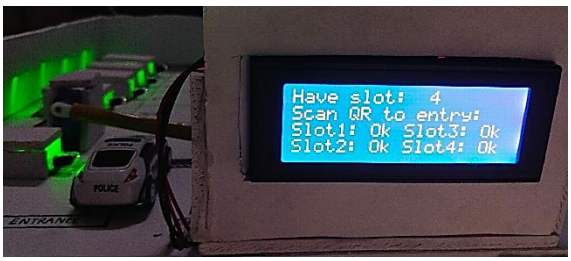
Hình 4-9: Trạng thái bãi đậu ban đầu

Sau khi đã kiểm tra tình trạng chỗ, trang đặt chỗ sẽ hiện ra khi chọn vào nút START BOOKING, lúc này ta sẽ thực hiện chọn các thông tin đặt chỗ ở giao diện, giả sử ở đây ta lựa chọn Slot1 tương ứng với vị trí số 1 với ngày đặt chỗ là 11-07-2024 và giờ vào là 02:07.



Hình 4-10: Thực hiện đặt chỗ

Một mã QR sẽ hiện ra khi ta nhấn vào nút Booking với bên dưới là các thông tin đặt chỗ đã chọn trước đó. Xe sẽ tiến hành chạy vào bãi đậu xe, ở cổng vào cảm biến hồng ngoại đã nhận diện có xe nên yêu cầu quét mã QR sẽ được hiện trên màn hình LCD.

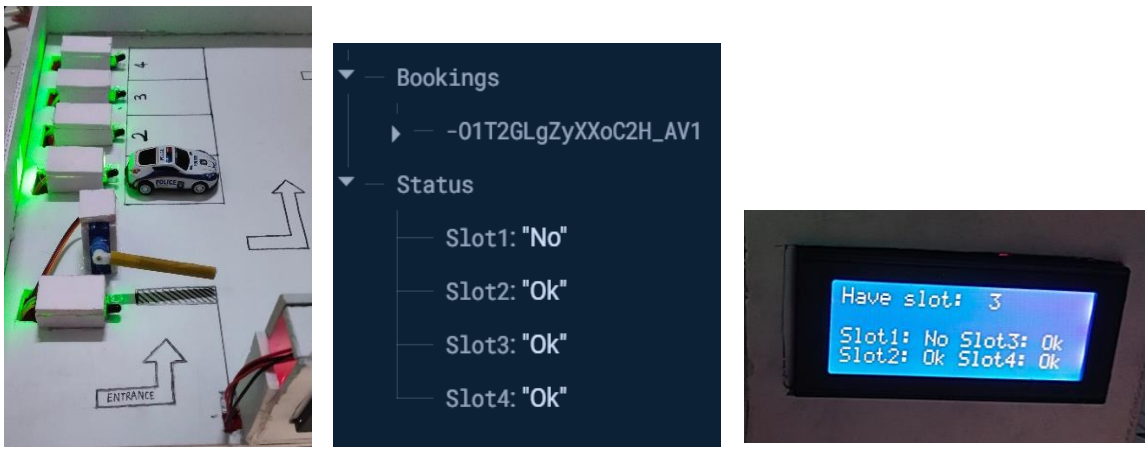


Hình 4-11: LCD yêu cầu quét mã QR

Cửa Servo sẽ mở ra cho xe vào bãi khi ta thực hiện quét mã QR hiển thị trên ứng dụng đặt chỗ.

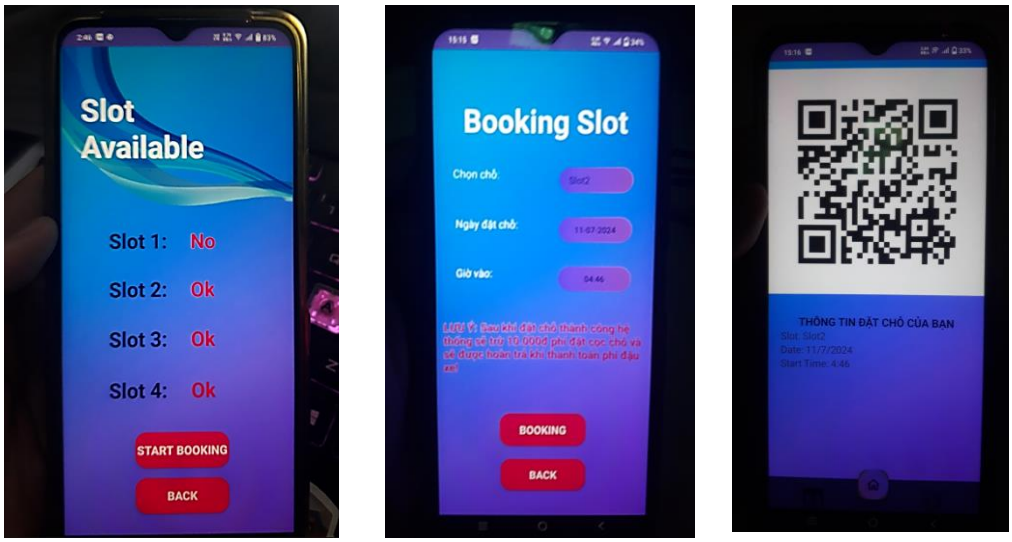
Hình 4-12: Thực hiện quét mã QR

Xe sẽ vào vị trí số 1 như đã đặt chỗ trước đó, trên màn hình LCD sẽ hiển thị còn trống 3 chỗ và vị trí số 1 trở thành "No" tức là vị trí Slot1 đã bị chiếm và chỉ còn trống 3 vị trí.



Hình 4-13: Trạng thái thay đổi khi xe thứ 1 vào

Người dùng khi quét mã QR sẽ được nhận diện là Slot1 nên hệ thống sẽ hiện thông báo trên màn hình chính để cho người dùng biết được mình đã vào đúng vị trí hay chưa để thực hiện điều chỉnh vị trí đậu xe. Chúng ta thực hiện đăng nhập trên một tài khoản khác và tiến hành các bước đặt chỗ tương tự như trên.



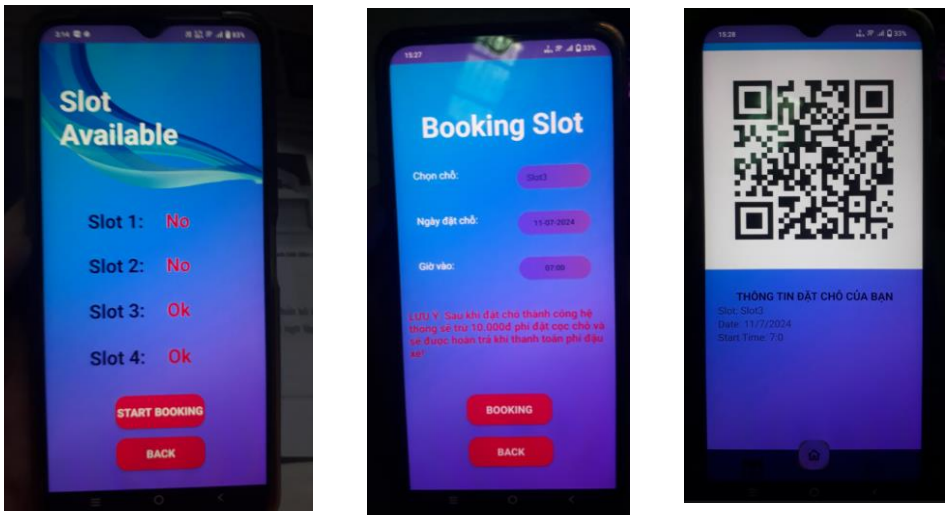
Hình 4-14: Giao diện đặt chỗ thay đổi khi xe thứ 1 vào

Chúng ta nhận thấy trên ứng dụng có xe đậu ở vị trí số 1 trước đó nên ta chỉ còn các chỗ 2, 3 và 4. Vị trí đậu số 2 sẽ được lựa chọn với ngày đặt chỗ là 11-07-2024 với giờ vào là 04:46, như vậy tiền đặt cọc của tài khoản người dùng sẽ tự động trừ đi 10000

đồng. Cảm biến hồng ngoại nhận diện có vật cản nên yêu cầu quét mã QR ở LCD khi ta cho xe vào hệ thống bãi đậu xe. Sau khi quét mã QR, ESP32 nhận biết được mã QR từ ứng dụng nên gửi tín hiệu đến servo mở cửa cho xe vào. Quá trình thực hiện hoàn toàn không xảy ra lỗi đạt với yêu cầu.

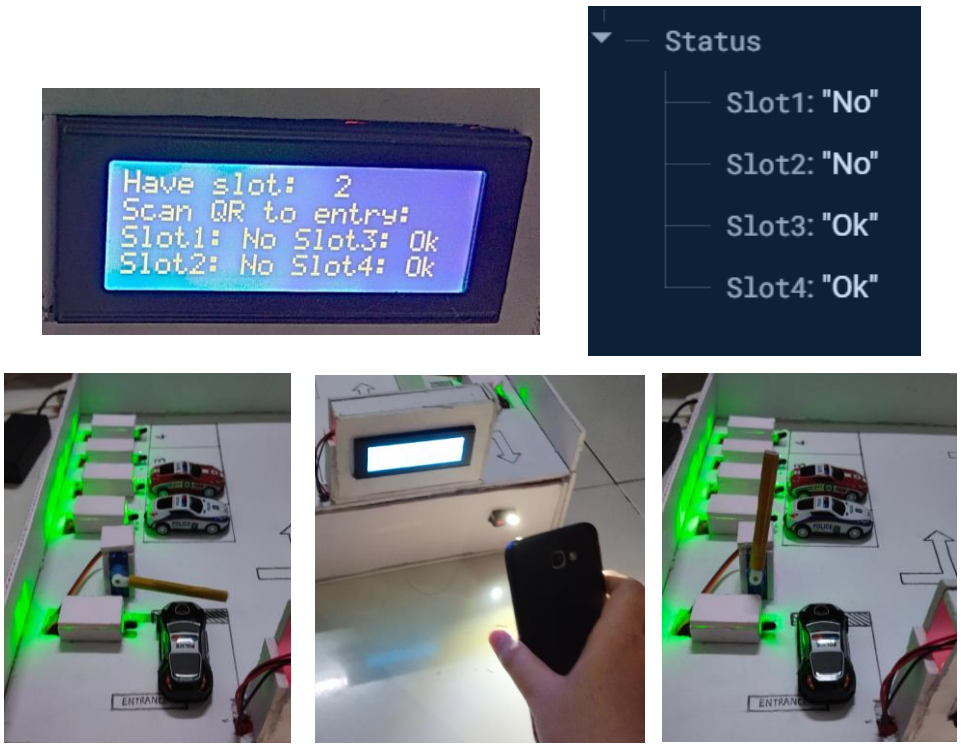
Hình 4-15: Trạng thái thay đổi khi xe thứ 2 vào

Các bước đặt chỗ được thực hiện tương tự cho xe thứ 3, ta nhận thấy ở Slot1 và Slot2 báo trạng thái "No" cho thấy có xe đậu vị trí số 1 và 2 chỉ còn lại 3 và 4 nên chỗ số 3 sẽ được lựa chọn với ngày đặt chỗ là 11-07-2024 và giờ vào là 07:00.



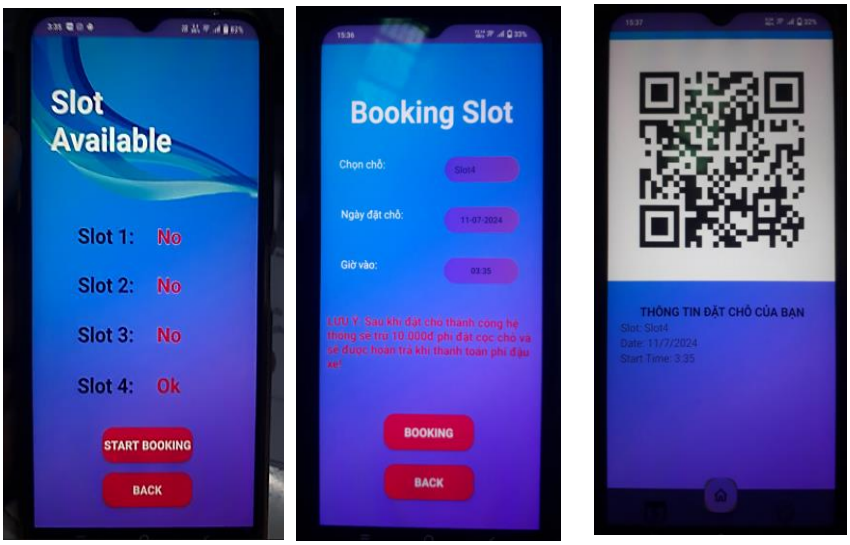
Hình 4-16: Giao diện đặt chỗ của xe thứ 3

Các bước thực hiện sẽ tương tự như trên, hệ thống nhận mã QR từ ứng dụng, cho xe vào và cập nhật thông tin lên cơ sở dữ liệu Firebase khi xe đã vào vị trí số 3.



Hình 4-17: Trạng thái hiển thị khi xe thứ 3 vào

Ta thực hiện việc đặt chỗ cho xe thứ 4 cũng là giới hạn của bãi đậu xe nên việc đặt chỗ chỉ còn lại ở vị trí số 4. Thực hiện việc đặt chỗ với lựa chọn cùng ngày như trên và giờ vào là 03:35, ta sẽ được mã QR cho vị trí số 4.

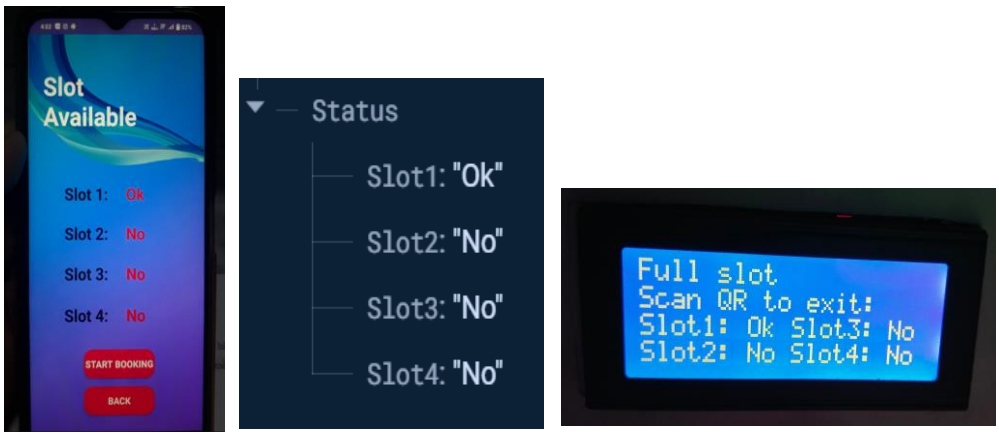


Hình 4-18: Giao diện đặt chỗ của xe thứ 4

Sau khi xe thứ 4 đã quét mã QR từ ứng dụng và vào trong bãi, trường hợp cả 4 xe đều đã đậu bên trong bãi sẽ xảy ra, màn hình LCD sẽ hiển thị ra thông báo “Full slot” cho người dùng biết rằng bãi đậu xe đã hết chỗ, đồng thời trên ứng dụng lẫn Firebase đều sẽ cập nhật trạng thái chỗ là “No”.

Hình 4-19: Trạng thái hiển thị khi xe thứ 4 vào bãi

Xe sẽ di chuyển ở lối ra của bãi khi người dùng muốn ra khỏi bãi. Sau khi xe chạy ra khỏi bãi đậu thì dữ liệu sẽ hiển thị trạng thái “Ok” ở vị trí số 1 trên cả LCD, ứng dụng lẫn Firebase.

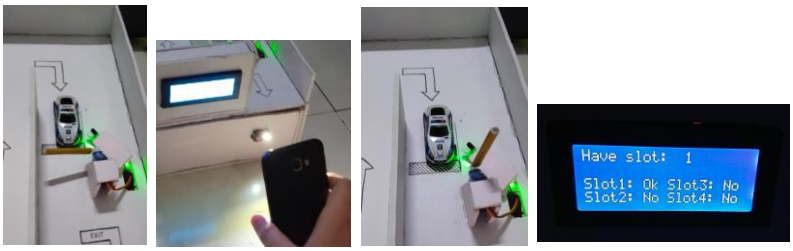


Hình 4-20: Trạng thái hiển thị khi xe thứ 1 ra khỏi bãi

Người dùng bấm vào nút Check Out tại giao diện màn hình chính của ứng dụng để ra khỏi bãi đậu xe. Một giao diện thanh toán được hiển thị với thời gian thực được ESP32 ghi nhận khi xe vào Slot1 và gửi lên hệ thống bao gồm thời gian vào, thời gian ra và giá tiền.

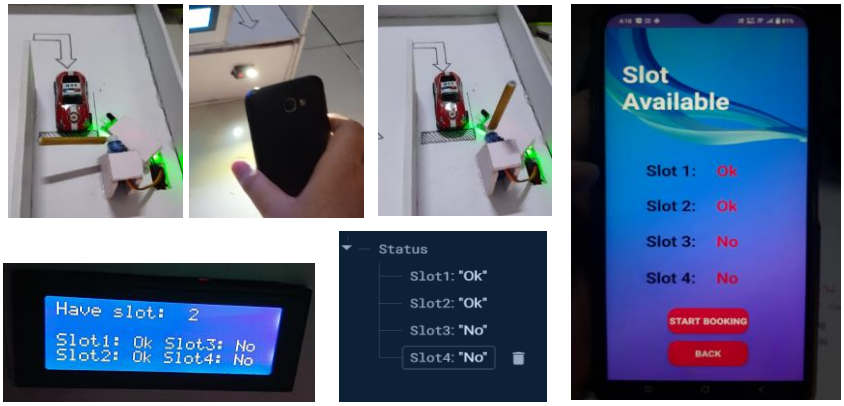
Hình 4-21: Thực hiện thanh toán và lấy mã QR

Khi đã thực hiện thanh toán thành công, người dùng sẽ nhận được mã QR để ra khỏi bãi xe. Sau khi quét mã QR, xe chạy ra khỏi bãi, LCD sẽ cập nhật trạng thái còn 1 vị trí trống lên màn hình.



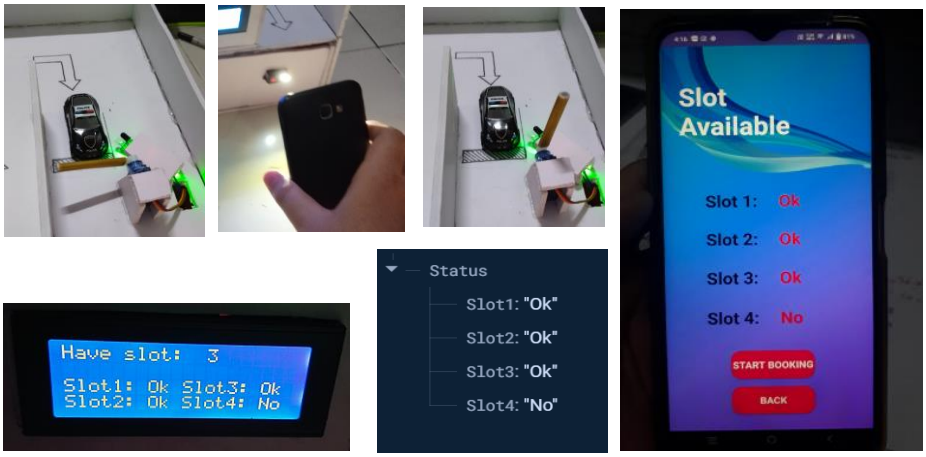
Hình 4-22: Trạng thái hiển thị khi xe thứ 1 ra khỏi bãi

Xe thứ 2 thực hiện các bước tương tự như xe thứ 1, người dùng sau khi quét mã QR và di chuyển xe ra khỏi bãi thì hệ thống sẽ cập nhật trạng thái còn 2 vị trí trống như mô tả ở Hình 4-23:



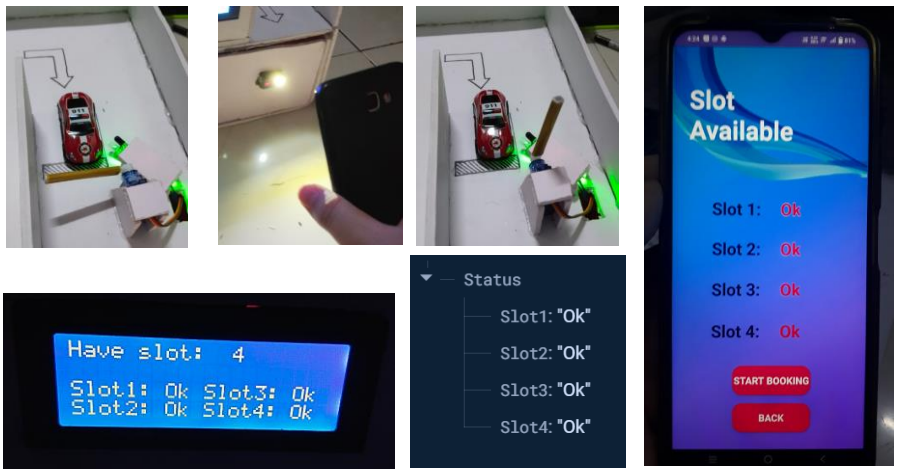
Hình 4-23: Trạng thái hiển thị khi xe thứ 2 ra khỏi bãi

Xe thứ 3 thực hiện các bước tương tự như trên, hệ thống cập nhật trạng thái còn lại 3 vị trí trống khi người dùng thực hiện thanh toán và quét mã QR để ra khỏi bãi như mô tả ở Hình 4-24:



Hình 4-25: Trạng thái hiển thị khi xe thứ 3 ra khỏi bãi

Xe thứ 4 sau khi ra khỏi bãi, trạng thái của bãi xe được cập nhật về 4 vị trí trống với trạng thái hiển thị tại giao diện ứng dụng là "Ok" như lúc đầu được miêu tả như Hình 4-25:



Hình 4-24: Trạng thái hiển thị khi xe thứ 4 ra khỏi bãi

4.2 Nhận xét, đánh giá

Sau khi thực hiện chạy thử mô hình thực tế, mô hình cho thấy được tính ổn định cao, các cảm biến hồng ngoại đều hoạt động với độ chính xác lên đến 98% và tỷ lệ sai số ở cảm biến hồng ngoại rơi vào khoảng 2% do gặp phải sự phản xạ của ánh sáng từ bề mặt không mong muốn. Servo hoạt động trơn tru, tốc độ phản hồi nhanh với độ chính

xác lên đến 99%, không bị kẹt hoặc nhiễu động khi điều khiển cửa của bãi đậu xe. Tỷ lệ sai số của Servo rơi vào khoảng 1% khi điều khiển góc quay trong một vài trường hợp nhưng không làm ảnh hưởng nhiều đến hệ thống. Màn hình LCD hiển thị chính xác và đầy đủ các thông tin về yêu cầu quét mã QR và trạng thái bãi đậu, không có tình trạng lỗi kí tự diễn ra với độ chính xác lên đến 99%. GM65 có thể đọc mã QR với độ chính xác và nhanh chóng lên đến 98% ở điều kiện lý tưởng, không nhận thấy tình trạng lỗi hoặc nhiễu dẫn đến không đọc được mã QR. Đối với phần mềm, ứng dụng cho tốc độ phản hồi lên đến 99% với độ trễ mạng không quá 1 giây khi cập nhật dữ liệu, không xảy ra các lỗi logic dẫn đến dừng ứng dụng hay bị sai lệch khi cập nhật tình hình bãi đậu. Giao tiếp truyền và nhận dữ liệu lên Firebase thường rất ổn định với tỷ lệ sai số dưới 1% trong điều kiện mạng ổn định. Trong một vài trường hợp điều kiện mạng không ổn định xảy ra do tốc độ mạng hoặc sự gián đoạn kết nối thì tỷ lệ này có thể tăng lên khoảng 10%. Nhìn chung, tỷ lệ sai số tổng thể của hệ thống có thể ước lượng khoảng 5 - 10% tùy thuộc vào điều kiện hoạt động và chất lượng của từng thành phần.

CHƯƠNG 5. KẾT LUẬN

5.1 Kết luận

Đề tài "Bãi đậu xe thông minh" về cơ bản đã đáp ứng được một số mục tiêu của nghiên cứu giúp người dùng giải quyết được bài toán tìm chỗ đậu xe, giúp tiết kiệm thời gian, bảo vệ môi trường, tối ưu hoá chi phí cho chủ đầu tư và hệ thống được triển khai một cách tự động nên sẽ hạn chế được tối đa sức lao động của người. Dưới đây là một số ưu điểm và khuyết điểm của hệ thống.

5.1.1 Ưu điểm

Hệ thống được thiết kế quản lý xe ra vào hoàn toàn tự động giúp giảm thiểu tối đa chi phí thuê nhân viên trông giữ tại bãi. Việc sử dụng mã QR tạo từ ứng dụng thay thế cho vé xé tay hay thẻ từ hoàn toàn phù hợp với xu hướng phát triển như hiện nay. Phí đậu xe sử dụng dữ liệu thời gian thực để tính toán và hiển thị một cách công khai, minh bạch giúp hạn chế tình trạng tăng giá bất thường như các bãi đậu xe truyền thống. Dữ liệu về tình trạng bãi đậu xe được cập nhật trực tiếp giúp người dùng có thể đưa ra sự lựa chọn đỗ xe mà không cần phải di chuyển liên tục để tìm bãi đậu xe giúp tiết kiệm thời gian và giảm lượng khí thải từ phương tiện ra môi trường. Hệ thống được thiết kế có giá thành thấp, rất phù hợp để triển khai nhân rộng hoặc nâng cấp thêm nhiều tính năng khác trong tương lai.

5.1.2 Khuyết điểm

Mặc dù hệ thống bãi đậu xe thông minh mang đến nhiều ưu điểm đáng chú ý nhưng bên cạnh đó hệ thống cũng còn tồn tại một số nhược điểm và thách thức. Một nhược điểm đáng chú ý nhất chính là giao diện ứng dụng còn quá đơn giản, chưa tích hợp được nhiều tính năng cho người dùng và chỉ sử dụng được trên hệ điều hành Android. Tính bảo mật của hệ thống còn chưa cao sẽ là mối nguy hại về việc lộ thông tin cá nhân người dùng. Việc đặt chỗ ở ứng dụng chưa thực sự hợp lí, cần phải điều chỉnh và nâng cấp thêm trong tương lai.

5.2 Hướng phát triển

Thông qua việc thực hiện đề tài này, một vài mặt hạn chế còn tồn đọng cần phải được cải thiện và phát triển thêm trong tương lai. Camera và cảm biến chuyển động cần phải được tích hợp thêm để tăng cường tính ổn định, thông minh và an ninh cho hệ thống. Ngoài ra, việc lập trình thêm tính năng nhận diện biển số và người lái xe sẽ giúp ứng dụng trở nên toàn diện hơn. Việc kết nối bãi đậu xe với các dịch vụ công cộng mang lại tiềm năng rất lớn nhằm cung cấp các thông tin hữu ích cho người dùng, chẳng hạn như dịch vụ rửa xe tự động, trạm sạc xe điện và hệ thống thông tin thời tiết. Để hướng đến việc sử dụng nguồn năng lượng sạch, đề tài cần kết hợp thêm các tấm pin sử dụng năng lượng mặt trời. Hơn nữa, mã QR tích hợp liên kết thanh toán tự động sẽ giúp người dùng thanh toán phí đậu xe một cách thuận tiện. Cải thiện khả năng tính toán giá tiền cho ứng dụng và nâng cao tính bảo mật cho mã QR là cần thiết để đảm bảo tính chính xác và bảo vệ dữ liệu không bị sai lệch hay sửa đổi.

Để mở rộng khả năng tiếp cận của ứng dụng, ứng dụng được phát triển thành một nền tảng di động đa hệ điều hành là cần thiết. Ngoài việc phát triển trên hệ điều hành Android, ứng dụng sẽ được mở rộng sang hệ điều hành iOS và thậm chí xây dựng một phiên bản web. Điều này sẽ giúp ứng dụng tiếp cận được với nhiều người dùng hơn, từ đó tăng khả năng sử dụng và sự tiện lợi cho khách hàng. Bên cạnh đó, việc tích hợp trí tuệ nhân tạo (AI) vào hệ thống cũng là một hướng phát triển quan trọng với xu hướng đô thị thông minh hiện nay. AI có thể phân tích dữ liệu để dự đoán nhu cầu đỗ xe, từ đó tối ưu hóa việc quản lý không gian đỗ xe một cách hiệu quả hơn. Ngoài ra, AI cũng có thể hỗ trợ trong việc nhận diện biển số xe và phát hiện các hành vi bất thường, giúp nâng cao tính an ninh và bảo mật cho hệ thống bãi đỗ xe thông minh.

TÀI LIỆU THAM KHẢO

Tiếng Việt

Vietnam Automobile Manufacturers' Association (VAMA), (11/06/2024), *Báo cáo bán hàng VAMA tháng 5 năm 2024* – Retrieved June 20, 2024 from website: http://vama.org.vn/vn/bao-cao-ban-hang.html

VINFAST, (17/04/2024), *VINFAST ghi nhận doanh thu tăng trưởng 269,7% trong quý 1/2024 so với cùng kỳ 2023* - Retrieved June 20, 2024 from website: https://vinfastauto.com/vn_vi/vinfast-ghi-nhan-doanh-thu-tang-truong-trong-quy-1-2024-so-voi-cung-ky-2023

Bùi Văn Minh, Dương Thanh Long, Phạm Quang Huy, (2019), *Lập Trình Điều Khiển Xa Với ESP8266-ESP32 Và ARDUINO*, NXB Thanh Niên.

Nguyễn Văn Hiệp, (2016), *Giáo trình lập trình Android trong ứng dụng điều khiển*, NXB Đại học Quốc gia Tp. Hồ Chí Minh.

Tiếng Anh

Kishankumar Gurjar, Nagendra Patahk, (August 28, 2018), *IoT Based Monitoring of Traffic and Car Parking in Smart cities*, LAP LAMBERT Academic Publishing.

Fikri Hanif, Mhd. Idham Khalif, (March, 2023), *Real Time Smart Car Parking System Using Internet of Things* - Retrieved June 20, 2024 from website: https://www.researchgate.net/publication/369305579_Real_Time_Smart_Car_Parking_System_Using_Internet_of_Things

Pratik Kasodariya, Shikha Patel, Dhaval H Rangrej, Hardik Tanti, (May 2020), *Smart Parking System based on IOT*, Retrieved June 20, 2024 from website: https://www.researchgate.net/publication/341870728_Smart_Parking_System_based_on_IOT

Arpit Atkare, Achal Bante, Shubhangi Gahane, Monika Ingole, Chandrashekhar Patle, *QR BASED CAR PARKING SYSTEM*, (2023), Retrieved June 20, 2024 from website: https://www.irjmets.com/uploadedfiles/paper//issue_12_december_2023/47293/final/fin_irjmets1702476852.pdf

PHỤ LỤC A MÃ NGUỒN ARDUINO

```
//Thu vien
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <ESP32Servo.h>
#include <WiFi.h>
#include <FirebaseESP32.h>
#include <NTPClient.h>
#include <WiFiUdp.h>

//Ket noi wifi
#define WIFI_SSID "Padoithong"
#define WIFI_PASSWORD "667788200801"

//Cau hinh Firebase
#define FIREBASE_HOST "baidoxethongminh-esp32-default-rtdb.asia-
southeast1.firebasedatabase.app/"
#define FIREBASE_AUTH
"OxwOsQJ3RURRP5TFonoYxQ2ZCfF0oUl0fg57oGSQ"

//Cac chan du lieu su dung
#define ir_enter 35
#define ir_exit 19
#define ir_car1 32
#define ir_car2 33
#define ir_car3 25
#define ir_car4 26

LiquidCrystal_I2C lcd(0x27, 20, 4);


Servo servoEnter;
Servo servoExit;

int temp = 0;
int slot = 4;

#define RXD2 16        // GPIO 16 => RX for uart
#define TXD2 17        // GPIO 17 => TX for uart
```

```
FirebaseData fbdb; //bien quan ly firebase

String qrcode;

String getSubstring(String data, String startDelimiter, String endDelimiter) {
  int startIndex = data.indexOf(startDelimiter);
  if (startIndex == -1) {
    return "";
  }
  startIndex += startDelimiter.length();
  int endIndex = data.indexOf(endDelimiter, startIndex);
  if (endIndex == -1) {
    return data.substring(startIndex);
  }
  return data.substring(startIndex, endIndex);
}

// Đặt múi giờ
const long utcOffsetInSeconds = 3600*7; // UTC+7

// Khởi tạo WiFi và NTP client
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds);

String data_entry, data_exit;
unsigned long entryTimes[4] = {0, 0, 0, 0};
bool slotOccupied[4] = {false, false, false, false};
int totalMoney;

String getCurrentTimeString() {
  int h = timeClient.getHours();
  int m = timeClient.getMinutes();
  int s = timeClient.getSeconds();
  return String(h) + ":" + String(m) + ":" + String(s);
}

void setup() {

  Serial2.begin(9600, SERIAL_8N1, RXD2, TXD2);
  Serial.begin (115200);
  servoEnter.attach(14);
  servoExit.attach(12);
```

```
  pinMode(ir_car1, INPUT);
  pinMode(ir_car2, INPUT);
  pinMode(ir_car3, INPUT);
  pinMode(ir_car4, INPUT);
  pinMode(ir_enter, INPUT);
  pinMode(ir_exit, INPUT);

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Welcome to");
  lcd.setCursor(0, 1);
  lcd.print("Smart Parking Place");
  delay(500);
  lcd.clear();

  Serial.print("Connecting to WIFI...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(5000);
    Serial.print(".");
  }
  //ket noi firebase database
  Firebase.begin(FIREBASE_HOST,FIREBASE_AUTH);
  Firebase.reconnectWiFi(true);

  Serial.print("Connected with IP: ");
  Serial.print(WiFi.localIP());
  Serial.println();

  // Khởi động NTP Client
  timeClient.begin();
  timeClient.update(); // Cập nhật thời gian lần đầu
}

void loop() {
  timeClient.update(); // Cập nhật thời gian thường xuyên
  if (slot > 0 && slot < 5)
    {
      lcd.setCursor(0, 0);
      lcd.print("Have slot: ");
      lcd.setCursor(12, 0);
```

```
    lcd.print(slot);
   }
  else
   {
     lcd.setCursor(0, 0);
     lcd.print("Full slot");
   }
   KiemTra();
   gate();
   time();
}

//Chuong trinh con
void gate()
{
   if (digitalRead(ir_enter) == 0)
   {
    lcd.setCursor(0, 1);
    lcd.print("Scan QR to entry:");
    MaQRvao();
   }
   if (slot < 4){
    if (digitalRead(ir_exit) == 0)
    {
     lcd.setCursor(0, 1);
     lcd.print("Scan QR to exit:");
     MaQRra();
    }
   }
}

void openDoorEnter()
{
   // Mở cửa vào
   servoEnter.write(90);
   delay(2000);
   // Đóng cửa vào
   servoEnter.write(0);
   delay(1000);
}

void openDoorExit()
{
```

```
   // Mở cửa ra
   servoExit.write(90);
   delay(2000);
   // Đóng cửa ra
   servoExit.write(0);
   delay(1000);
}

void MaQRvao()
{
  while (Serial2.available())
  {
    qrcode = Serial2.readString();
    Serial.println(qrcode);
    String slotString = getSubstring(qrcode, "Slot: ", "\n");
    Serial.println("Gia tri Slot da cat: '" + slotString + "'"); // In ra giá trị slot đã
cắt
       if (slotString == "Slot1" || slotString == "Slot2" || slotString == "Slot3" ||
slotString == "Slot4")
       {
          Serial.println("moi xe vao");
          openDoorEnter();
          slot--;
          String basePath = "/Time/" + slotString;
          Firebase.setInt(fbdb, basePath + "/Trangthai", 0);
          lcd.clear();
       } else {
     Serial.println("Khong hop le");}
  }
}


void MaQRra()
{
  while (Serial2.available())
  {
    qrcode = Serial2.readString();
    Serial.println(qrcode);
    String slotString = getSubstring(qrcode, "Slot: ", "\n");
    Serial.println("Gia tri Slot da cat: '" + slotString + "'"); // In ra giá trị slot đã
cắt
       if (slotString == "Slot1" || slotString == "Slot2" || slotString == "Slot3" ||
slotString == "Slot4")
```

```
      {
        Serial.println("moi xe ra");
        openDoorExit();
        slot++;
        String basePath = "/Time/" + slotString;
        Firebase.setInt(fbdb, basePath + "/Trangthai", 1);
        lcd.clear();
      } else {
    Serial.println("Khong hop le");}
 }
}

void KiemTra()
{
 if(Firebase.getInt(fbdb, "Status/Slot1") == true)
 {

  if (digitalRead(ir_car1) == 0)
  {
   lcd.setCursor(0, 2);
   lcd.print("Slot1: No");
   Firebase.setString(fbdb, "Status/Slot1","No");
  }
  else
  {
   lcd.setCursor(0, 2);
   lcd.print("Slot1: Ok");
   Firebase.setString(fbdb, "Status/Slot1","Ok");
  }
 }

 if(Firebase.getInt(fbdb, "Status/Slot2" ) == true)
 {

  if (digitalRead(ir_car2) == 0)
  {
   lcd.setCursor(0, 3);
   lcd.print("Slot2: No");
   Firebase.setString(fbdb, "Status/Slot2","No");
  }
  else
  {
   lcd.setCursor(0, 3);
```

```
    lcd.print("Slot2: Ok");
    Firebase.setString(fbdb, "Status/Slot2","Ok");
   }
  }

  if(Firebase.getInt(fbdb, "Status/Slot3" ) == true)
  {

   if (digitalRead(ir_car3) == 0)
   {
   lcd.setCursor(10, 2);
   lcd.print("Slot3: No");
   Firebase.setString(fbdb, "Status/Slot3","No");
   }
   else
   {
   lcd.setCursor(10, 2);
   lcd.print("Slot3: Ok");
   Firebase.setString(fbdb, "Status/Slot3","Ok");
   }
  }

  if(Firebase.getInt(fbdb, "Status/Slot4" ) == true)
  {

   if (digitalRead(ir_car4) == 0)
   {
    lcd.setCursor(10, 3);
    lcd.print("Slot4: No");
    Firebase.setString(fbdb, "Status/Slot4","No");
   }
   else
   {
    lcd.setCursor(10, 3);
    lcd.print("Slot4: Ok");
    Firebase.setString(fbdb, "Status/Slot4","Ok");
   }
  }
}

void time()
{
  // Mảng cảm biến
```

```
  int sensors[] = {ir_car1, ir_car2, ir_car3, ir_car4};

  for (int i = 0; i < 4; i++) {
    if (digitalRead(sensors[i]) == 0) {
      // Xe vào
      if (!slotOccupied[i]) {
        data_entry = getCurrentTimeString();
        Serial.println(data_entry);
        entryTimes[i] = millis();
        slotOccupied[i] = true;
        Firebase.setString(fbdb, String("Time/Slot") + (i + 1) + "/Entry",
data_entry);
      }
    } else {
      // Xe ra
      if (slotOccupied[i]) {
        data_exit = getCurrentTimeString();
        Serial.println(data_exit);
        unsigned long elapsedTime = millis() - entryTimes[i];
        entryTimes[i] = 0;
        slotOccupied[i] = false;
        totalMoney = tinhtien(elapsedTime);
        Firebase.setString(fbdb, String("Time/Slot") + (i + 1) + "/Exit", data_exit);
        Firebase.setString(fbdb, String("Time/Slot") + (i + 1) + "/Total",
String(totalMoney));
      }
    }
  }
}

int tinhtien(unsigned long elapsedTime)
{
  if (elapsedTime < 7200000) { // Ít hơn 2 giờ
    return 20000;
  } else if (elapsedTime >= 7200000 && elapsedTime < 10800000) { // Từ 2 giờ
đến dưới 3 giờ
    return 30000;
  } else { // Từ 3 giờ trở lên
    return 50000;
  }
}
```

PHỤ LỤC B MÃ NGUỒN ANDROID STUDIO

Tệp LoginActivity.Java

```
package com.iot.baidoxethongminh.Activity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.iot.baidoxethongminh.Adapter.RegisterActivity;
import com.iot.baidoxethongminh.R;

public class LoginActivity extends AppCompatActivity {
    private EditText emailEdt, passEdt;
    private Button Loginbtn, registerBtn;
    private FirebaseAuth mAuth;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);

        mAuth = FirebaseAuth.getInstance();

        initView();
        setVariable();
    }
    private void initView(){
        emailEdt=findViewById(R.id.emailtext);
        passEdt=findViewById(R.id.passtext);
        Loginbtn=findViewById(R.id.Loginbtn);
```

```
        registerBtn = findViewById(R.id.registerBtn);
    }

    private void setVariable() {

        Loginbtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                String email = emailEdt.getText().toString();
                String password = passEdt.getText().toString();

                if (email.isEmpty() || password.isEmpty()) {
                    Toast.makeText(LoginActivity.this, "Please fill the login form",
Toast.LENGTH_SHORT).show();
                } else {
                    mAuth.signInWithEmailAndPassword(email, password)
                            .addOnCompleteListener(LoginActivity.this, new
OnCompleteListener<AuthResult>() {
                                @Override
                                public void onComplete(@NonNull Task<AuthResult> task)
{
                                    if (task.isSuccessful()) {
                                        startActivity(new Intent(LoginActivity.this,
MainActivity.class));
                                        finish();
                                    } else {
                                        Toast.makeText(LoginActivity.this, "Authentication
Failed: " + task.getException().getMessage(), Toast.LENGTH_SHORT).show();
                                    }
                                }
                            });
                }
            }
        });

        registerBtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(LoginActivity.this, RegisterActivity.class));
            }
        });
    }
}
```

Tệp RegisterActivity.Java

```
package com.iot.baidoxethongminh.Adapter;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.iot.baidoxethongminh.Activity.LoginActivity;
import com.iot.baidoxethongminh.R;

public class RegisterActivity extends AppCompatActivity {

    private EditText emailEdt, passEdt;
    private Button registerBtn, backBtn;
    private FirebaseAuth mAuth;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_register);

        mAuth = FirebaseAuth.getInstance();

        emailEdt = findViewById(R.id.editTextEmail);
        passEdt = findViewById(R.id.editTextPassword);
        registerBtn = findViewById(R.id.registerButton);
        backBtn = findViewById(R.id.backButton);

        registerBtn.setOnClickListener(new View.OnClickListener() {
            @Override
```

```
        public void onClick(View v) {
            String email = emailEdt.getText().toString();
            String password = passEdt.getText().toString();

            if (email.isEmpty() || password.isEmpty()) {
                Toast.makeText(RegisterActivity.this, "Please fill the registration
form", Toast.LENGTH_SHORT).show();
                return;
            }

            mAuth.createUserWithEmailAndPassword(email, password)
                    .addOnCompleteListener(RegisterActivity.this, new
OnCompleteListener<AuthResult>() {
                        @Override
                        public void onComplete(@NonNull Task<AuthResult> task) {
                            if (task.isSuccessful()) {
                                Toast.makeText(RegisterActivity.this, "Registration
Successful", Toast.LENGTH_SHORT).show();
                                finish(); // Go back to the login activity
                            } else {
                                Toast.makeText(RegisterActivity.this, "Registration Failed:
" + task.getException().getMessage(), Toast.LENGTH_SHORT).show();
                            }
                        }
                    });
        }
    });
    backBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(RegisterActivity.this, LoginActivity.class));
        }
    });
  }
}
```

Tệp MainActivity.java

```
package com.iot.baidoxethongminh.Activity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
```

```
import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;

import
com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.firebase.auth.FirebaseAuth;
import com.iot.baidoxethongminh.Adapter.UserBookingsActivity;
import com.iot.baidoxethongminh.R;

public class MainActivity extends AppCompatActivity {
    private Button booking, btnViewBookings, logoutBtn, fastparkingbtn;
    private FloatingActionButton fab;
    private FirebaseAuth mAuth;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mAuth = FirebaseAuth.getInstance();

        initView();
        setVariable();
        viewslot();
    }
    private void initView(){
        booking=findViewById(R.id.findparking);
        btnViewBookings = findViewById(R.id.btnViewBookings);
        fab = findViewById(R.id.fab);
        logoutBtn = findViewById(R.id.logoutBtn);
        fastparkingbtn = findViewById(R.id.aboutbtn);
    }
    private void setVariable(){
        booking.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(MainActivity.this, MapsActivity.class));
            }
        });
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finishAffinity();
```

```
            System.exit(0);
        }
    });
    logoutBtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mAuth.signOut();
            startActivity(new Intent(MainActivity.this, LoginActivity.class));
            finish();
        }
    });
    fastparkingbtn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(MainActivity.this, AboutActivity.class));
        }
    });
}

private void viewslot(){
    btnViewBookings.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(MainActivity.this,
UserBookingsActivity.class);
            startActivity(intent);
        }
    });
}
}
```

Tệp MapsActivity.java

```
package com.iot.baidoxethongminh.Activity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import androidx.activity.EdgeToEdge;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
```

```
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.iot.baidoxethongminh.R;

import java.util.zip.Inflater;

public class MapsActivity extends AppCompatActivity implements
OnMapReadyCallback {
   private GoogleMap mMap;
   private Button btnChangeLayout, back;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_maps);

      // Obtain the SupportMapFragment and get notified when the map is ready
to be used.
      SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
      mapFragment.getMapAsync(this);

      btnChangeLayout = findViewById(R.id.btnChangeLayout);
      back = findViewById(R.id.back);
      btnChangeLayout.setVisibility(View.GONE); // Ẩn nút btnChangeLayout
ban đầu
      back.setVisibility(View.VISIBLE); // Hiển thị nút back ban đầu

      btnChangeLayout.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            Intent intent = new Intent(MapsActivity.this, DetailActivity.class);
            startActivity(intent);
         }
      });
```

```
        back.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MapsActivity.this, MainActivity.class);
                startActivity(intent);
            }
        });
    }

    @Override
    public void onMapReady(@NonNull GoogleMap googleMap) {
        mMap = googleMap;

        // Add markers for parking locations
        LatLng parking1 = new LatLng(10.733609684695706,
106.69989264603498); //TDTU
        LatLng parking2 = new LatLng(10.741145138461116,
106.70178711219728); //Lotte Q7
        LatLng parking3 = new LatLng(10.758677894379538,
106.7125631109268); //Home

        mMap.addMarker(new MarkerOptions().position(parking1).title("Parking
1"));
        mMap.addMarker(new MarkerOptions().position(parking2).title("Parking
2"));
        mMap.addMarker(new MarkerOptions().position(parking3).title("Parking
3"));

        // Move the camera to the first parking location
        mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(parking1,
14));

        mMap.setOnMarkerClickListener(new
GoogleMap.OnMarkerClickListener() {
            @Override
            public boolean onMarkerClick(Marker marker) {
                // Hiển thị nút btnChangeLayout khi nhấn vào marker và ẩn nút back
                btnChangeLayout.setVisibility(View.VISIBLE);
                back.setVisibility(View.GONE);
                return false;
            }
        });
```

```
        // Đặt sự kiện để ẩn btnChangeLayout và hiển thị lại back khi nhấp vào bản
đồ
        mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
            @Override
            public void onMapClick(LatLng latLng) {
                btnChangeLayout.setVisibility(View.GONE);
                back.setVisibility(View.VISIBLE);
            }
```

Tệp DetailActivity.java

```
package com.iot.baidoxethongminh.Activity;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;


import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.iot.baidoxethongminh.R;

public class DetailActivity extends AppCompatActivity {
    private TextView tv1, tv2, tv3, tv4;
    private Button btb1, btb2;
    private static final String TAG = "DetailActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detail);
```

```
    btb1 = findViewById(R.id.btb1);
    btb2 = findViewById(R.id.btb2);
    tv1 = findViewById(R.id.tv1);
    tv2 = findViewById(R.id.tv2);
    tv3 = findViewById(R.id.tv3);
    tv4 = findViewById(R.id.tv4);
    setEvent();
    setEvent1();
    setEvent2();
    setEvent3();
    setVariable();
    setVariable2();
}

private void setEvent() {
    FirebaseDatabase database = FirebaseDatabase.getInstance();
    DatabaseReference myRef = database.getReference("Status/Slot1");

    myRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            String value = snapshot.getValue(String.class);
            Log.d(TAG, "Value is: " + value);
            tv1.setText(value);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.w(TAG, "Failed to read value.", error.toException());
        }
    });
}

private void setEvent1() {
    FirebaseDatabase database = FirebaseDatabase.getInstance();
    DatabaseReference myRef = database.getReference("Status/Slot2");

    myRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            String value = snapshot.getValue(String.class);
            Log.d(TAG, "Value is: " + value);
```

```
            tv2.setText(value);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.w(TAG, "Failed to read value.", error.toException());
        }
    });
}

private void setEvent2() {
    FirebaseDatabase database = FirebaseDatabase.getInstance();
    DatabaseReference myRef = database.getReference("Status/Slot3");

    myRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            String value = snapshot.getValue(String.class);
            Log.d(TAG, "Value is: " + value);
            tv3.setText(value);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Log.w(TAG, "Failed to read value.", error.toException());
        }
    });
}

private void setEvent3() {
    FirebaseDatabase database = FirebaseDatabase.getInstance();
    DatabaseReference myRef = database.getReference("Status/Slot4");

    myRef.addValueEventListener(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            String value = snapshot.getValue(String.class);
            Log.d(TAG, "Value is: " + value);
            tv4.setText(value);
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
```

```
            Log.w(TAG, "Failed to read value.", error.toException());
        }
    });
}

private void setVariable() {
    btb1.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(DetailActivity.this, BookingActivity.class));
        }
    });
}

private void setVariable2() {
    btb2.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(DetailActivity.this, MapsActivity.class));
        }
    });
}
}
```

Tệp BookingActivity.java

```
package com.iot.baidoxethongminh.Activity;

import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
import androidx.annotation.NonNull;
```

```
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.android.gms.tasks.TaskCompletionSource;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.MutableData;
import com.google.firebase.database.Transaction;
import com.google.firebase.database.ValueEventListener;
import com.google.firebase.database.annotations.Nullable;
import com.iot.baidoxethongminh.Adapter.BookingInfo;
import com.iot.baidoxethongminh.R;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;

public class BookingActivity extends AppCompatActivity {
    private Spinner spinnerSlot;
    private Button btnSelectDate, btnSelectStartTime, btnReserveAndPay,
btnback;
    private String selectedSlot;
    private Calendar selectedDate, startTime;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_booking);

        spinnerSlot = findViewById(R.id.spinnerSlot);
        btnSelectDate = findViewById(R.id.btnSelectDate);
        btnSelectStartTime = findViewById(R.id.btnSelectStartTime);
```

```
        btnReserveAndPay = findViewById(R.id.btnReserveAndPay);
        btnback = findViewById(R.id.btnback);

        SlotsFromFirebase();

        btnback.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivities(new Intent[]{new Intent(BookingActivity.this,
DetailActivity.class)});
            }
        });

        btnSelectDate.setOnClickListener(v -> {
            final Calendar c = Calendar.getInstance();
            int year = c.get(Calendar.YEAR);
            int month = c.get(Calendar.MONTH);
            int day = c.get(Calendar.DAY_OF_MONTH);

            DatePickerDialog datePickerDialog = new
DatePickerDialog(BookingActivity.this,
                    (view, year1, monthOfYear, dayOfMonth) -> {
                        selectedDate = Calendar.getInstance();
                        selectedDate.set(year1, monthOfYear, dayOfMonth);
                        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy",
Locale.getDefault());
                        btnSelectDate.setText(sdf.format(selectedDate.getTime()));
                    }, year, month, day);
            datePickerDialog.show();
        });

        btnSelectStartTime.setOnClickListener(v -> {
            final Calendar c = Calendar.getInstance();
            int hour = c.get(Calendar.HOUR_OF_DAY);
            int minute = c.get(Calendar.MINUTE);

            TimePickerDialog timePickerDialog = new
TimePickerDialog(BookingActivity.this,
                    (view, hourOfDay, minute1) -> {
                        startTime = Calendar.getInstance();
                        startTime.set(Calendar.HOUR_OF_DAY, hourOfDay);
                        startTime.set(Calendar.MINUTE, minute1);
```

```
                SimpleDateFormat sdf = new SimpleDateFormat("HH:mm",
Locale.getDefault());
                btnSelectStartTime.setText(sdf.format(startTime.getTime()));
            }, hour, minute, true);
        timePickerDialog.show();
    });

    btnReserveAndPay.setOnClickListener(v -> {
        if (selectedSlot != null && selectedDate != null && startTime != null) {
            // Kiểm tra trùng lặp đặt chỗ
            checkForDuplicateBooking(selectedSlot, selectedDate, startTime, new
OnCompleteListener<Boolean>() {
                @Override
                public void onComplete(@NonNull Task<Boolean> task) {
                    if (task.isSuccessful()) {
                        boolean isDuplicate = task.getResult();
                        if (isDuplicate) {
                            // Thông báo cho người dùng rằng đặt chỗ trùng lặp
                            Toast.makeText(BookingActivity.this, "Slot này đã được đặt
chỗ cho thời gian đã chọn. Vui lòng chọn thời gian khác.",
Toast.LENGTH_SHORT).show();
                        } else {
                            // Lấy thông tin người dùng hiện tại
                            String userId =
FirebaseAuth.getInstance().getCurrentUser().getUid();
                            DatabaseReference userRef =
FirebaseDatabase.getInstance().getReference("users").child(userId);

                            // Kiểm tra số dư của người dùng
                            userRef.child("balance").get().addOnCompleteListener(task1
-> {
                                if (task1.isSuccessful() && task1.getResult().exists()) {
                                    long currentBalance =
task1.getResult().getValue(Long.class);

                                    // Kiểm tra số dư có đủ để trừ không
                                    if (currentBalance >= 10000) {
                                        // Trừ 10000 vào balance
                                        userRef.child("balance").setValue(currentBalance -
10000).addOnCompleteListener(balanceUpdateTask -> {
                                            if (balanceUpdateTask.isSuccessful()) {
                                                // Tiến hành lưu thông tin đặt chỗ
```

```
                                DatabaseReference bookingsRef =
FirebaseDatabase.getInstance().getReference("Bookings");
                                String bookingId = bookingsRef.push().getKey();
                                String date =
selectedDate.get(Calendar.DAY_OF_MONTH) + "/" +
(selectedDate.get(Calendar.MONTH) + 1) + "/" +
selectedDate.get(Calendar.YEAR);
                                String startTimeStr =
startTime.get(Calendar.HOUR_OF_DAY) + ":" +
startTime.get(Calendar.MINUTE);

                                BookingInfo bookingInfo = new
BookingInfo(selectedSlot, date, startTimeStr, userId);

bookingsRef.child(bookingId).setValue(bookingInfo)
                                        .addOnCompleteListener(bookingTask -> {
                                            if (bookingTask.isSuccessful()) {
                                                // Chuyển tiếp đến InfoActivity nếu đặt
chỗ thành công
                                                Intent intent = new
Intent(BookingActivity.this, InfoActivity.class);
                                                intent.putExtra("reservationDetails",
"Slot: " + bookingInfo.slot + "\n" + "Date: " + bookingInfo.date + "\n" + "Start
Time: " + bookingInfo.startTime);
                                                intent.putExtra("bookingInfo",
bookingInfo); // Truyền đối tượng BookingInfo
                                                startActivity(intent);
                                            } else {
                                                Toast.makeText(BookingActivity.this,
"Đặt chỗ thất bại. Vui lòng thử lại.", Toast.LENGTH_SHORT).show();
                                            }
                                        });
                                } else {
                                    Toast.makeText(BookingActivity.this, "Cập nhật
số dư thất bại. Vui lòng thử lại.", Toast.LENGTH_SHORT).show();
                                }
                            });
                        } else {
                            Toast.makeText(BookingActivity.this, "Số dư không
đủ để đặt chỗ.", Toast.LENGTH_SHORT).show();
                        }
                    } else {
```

```
                                Toast.makeText(BookingActivity.this, "Không thể lấy
thông tin số dư. Vui lòng thử lại.", Toast.LENGTH_SHORT).show();
                            }
                        });
                    }
                } else {
                    Toast.makeText(BookingActivity.this, "Có lỗi xảy ra khi kiểm
tra trùng lặp. Vui lòng thử lại.", Toast.LENGTH_SHORT).show();
                }
            }
        });
    } else {
        Toast.makeText(BookingActivity.this, "Vui lòng điền đầy đủ thông
tin.", Toast.LENGTH_LONG).show();
    }
  });
}

  private void SlotsFromFirebase() {
    DatabaseReference slotsRef =
FirebaseDatabase.getInstance().getReference("Status");
    slotsRef.addValueEventListener(new ValueEventListener() {
      @Override
      public void onDataChange(@NonNull DataSnapshot snapshot) {
        List<String> availableSlots = new ArrayList<>();
        for (DataSnapshot slotSnapshot : snapshot.getChildren()) {
          String slotId = slotSnapshot.getKey();
          String status = slotSnapshot.getValue(String.class);
          if ("Ok".equals(status)) {
            availableSlots.add(slotId);
          }
        }
        ArrayAdapter<String> adapter = new
ArrayAdapter<>(BookingActivity.this,
android.R.layout.simple_spinner_dropdown_item, availableSlots);
        spinnerSlot.setAdapter(adapter);
      }

      @Override
      public void onCancelled(@NonNull DatabaseError error) {
        Toast.makeText(BookingActivity.this, "Failed to load slots.",
Toast.LENGTH_SHORT).show();
      }
```

```
        });

        spinnerSlot.setOnItemSelectedListener(new
AdapterView.OnItemSelectedListener() {
            @Override
            public void onItemSelected(AdapterView<?> parent, View view, int
position, long id) {
                selectedSlot = (String) parent.getItemAtPosition(position);
            }

            @Override
            public void onNothingSelected(AdapterView<?> parent) {
                selectedSlot = null;
            }
        });
    }

    private void checkForDuplicateBooking(String slot, Calendar date, Calendar
startTime, OnCompleteListener<Boolean> onCompleteListener) {
        DatabaseReference bookingsRef =
FirebaseDatabase.getInstance().getReference("Bookings");

        String selectedDateStr = date.get(Calendar.DAY_OF_MONTH) + "/" +
(date.get(Calendar.MONTH) + 1) + "/" + date.get(Calendar.YEAR);
        String selectedStartTimeStr = startTime.get(Calendar.HOUR_OF_DAY) +
":" + startTime.get(Calendar.MINUTE);

        bookingsRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                boolean isDuplicate = false;
                for (DataSnapshot bookingSnapshot : snapshot.getChildren()) {
                    BookingInfo bookingInfo =
bookingSnapshot.getValue(BookingInfo.class);

                    if (bookingInfo != null && bookingInfo.slot.equals(slot)
                            && bookingInfo.date.equals(selectedDateStr)
                            && bookingInfo.startTime.equals(selectedStartTimeStr)) {
                        isDuplicate = true;
                        break;
                    }
                }
```

```
            TaskCompletionSource<Boolean> taskCompletionSource = new
TaskCompletionSource<>();
            taskCompletionSource.setResult(isDuplicate);
            onCompleteListener.onComplete(taskCompletionSource.getTask());
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            TaskCompletionSource<Boolean> taskCompletionSource = new
TaskCompletionSource<>();
            taskCompletionSource.setResult(false);
            onCompleteListener.onComplete(taskCompletionSource.getTask());
        }
    });
  }
}
```

Tệp BookingInfo.java

```
package com.iot.baidoxethongminh.Adapter;

import java.io.Serializable;

public class BookingInfo implements Serializable {
  public String slot;
  public String date;
  public String startTime;
  public String userId;

  public BookingInfo() {
  }

  public BookingInfo(String slot, String date, String startTime, String userId) {
    this.slot = slot;
    this.date = date;
    this.startTime = startTime;
    this.userId = userId;
  }
}
```

Tệp InfoActivity.java

```
package com.iot.baidoxethongminh.Activity;
```

```
import android.content.Intent;
import android.graphics.Bitmap;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import
com.google.android.material.floatingactionbutton.FloatingActionButton;
import com.google.zxing.BarcodeFormat;
import com.iot.baidoxethongminh.Adapter.BookingInfo;
import com.iot.baidoxethongminh.Adapter.Notifications;
import com.iot.baidoxethongminh.R;
import com.journeyapps.barcodescanner.BarcodeEncoder;

public class InfoActivity extends AppCompatActivity {
    private ImageView ivQrCode;
    private TextView tvSlot, tvDate, tvStartTime;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_info);

        ivQrCode = findViewById(R.id.ivQrCode);
        tvSlot = findViewById(R.id.tvSlot);
        tvDate = findViewById(R.id.tvDate);
        tvStartTime = findViewById(R.id.tvStartTime);
        FloatingActionButton fab = findViewById(R.id.fab);


        Intent intent = getIntent();
        String reservationDetails = intent.getStringExtra("reservationDetails");


        try {
            BarcodeEncoder barcodeEncoder = new BarcodeEncoder();
            Bitmap bitmap = barcodeEncoder.encodeBitmap(reservationDetails,
BarcodeFormat.QR_CODE, 400, 400);
            ivQrCode.setImageBitmap(bitmap);
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }

        // Lấy thông tin đặt chỗ từ intent và hiển thị
        BookingInfo bookingInfo = (BookingInfo)
intent.getSerializableExtra("bookingInfo");
        if (bookingInfo != null) {
            tvSlot.setText("Slot: " + bookingInfo.slot);
            tvDate.setText("Date: " + bookingInfo.date);
            tvStartTime.setText("Start Time: " + bookingInfo.startTime);
        }

        fab.setOnClickListener(view -> {
            Intent back = new Intent(InfoActivity.this, MainActivity.class);
            startActivity(back);
            finish();
        });
    }
}
```

Tệp PaymentActivity.java

```
package com.iot.baidoxethongminh.Activity;

import android.content.Intent;

import android.os.Bundle;

import android.util.Log;

import android.view.View;

import android.widget.Button;

import android.widget.TextView;

import android.widget.Toast;


import androidx.annotation.NonNull;

import androidx.appcompat.app.AppCompatActivity;


import com.google.firebase.auth.FirebaseAuth;
```

```
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.iot.baidoxethongminh.Adapter.UserBookingsActivity;
import com.iot.baidoxethongminh.R;

public class PaymentActivity extends AppCompatActivity {

    private TextView tvSlot, tvEntryTime, tvExitTime, tvTotalAmount, tvDiscountedAmount;
    private Button btnPay, btnback;
    private int discountedAmount;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_payment);

        // Ánh xạ các View từ giao diện
        tvSlot = findViewById(R.id.tvSlot);
        tvEntryTime = findViewById(R.id.tvEntryTime);
        tvExitTime = findViewById(R.id.tvExitTime);
        tvTotalAmount = findViewById(R.id.tvTotalAmount);
        tvDiscountedAmount = findViewById(R.id.tvDiscountedAmount);
        btnPay = findViewById(R.id.btnPay);
        btnback = findViewById(R.id.btnback);
```

```
        // Khởi tạo và bắt đầu theo dõi trạng thái của các slot
        monitorFirebaseForSlotChanges();

        btnback.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startActivity(new Intent(PaymentActivity.this, MainActivity.class));
            }
        });

        // Thiết lập sự kiện nhấn nút "Pay"
        btnPay.setOnClickListener(view -> processPayment());
    }

    // Phương thức theo dõi thay đổi trạng thái của các slot trên Firebase
    private void monitorFirebaseForSlotChanges() {
        for (int i = 1; i <= 4; i++) {
            String slotId = "Slot" + i;
            DatabaseReference timeSlotRef =
FirebaseDatabase.getInstance().getReference("Time").child(slotId).child("Trangt
hai");
            DatabaseReference slotStatusRef =
FirebaseDatabase.getInstance().getReference("Status").child(slotId);

            timeSlotRef.addValueEventListener(new ValueEventListener() {
                @Override
                public void onDataChange(@NonNull DataSnapshot timeSnapshot) {
                    Integer trangthai = timeSnapshot.getValue(Integer.class);
```

```
            if (trangthai != null && trangthai == 0) {
                slotStatusRef.addValueEventListener(new ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot statusSnapshot) {
                        String status = statusSnapshot.getValue(String.class);

                        if ("Ok".equals(status)) {
                            tvSlot.setText("Slot: " + slotId);
                            fetchPaymentDetails(slotId);
                        }
                    }
                    @Override
                    public void onCancelled(@NonNull DatabaseError error) {
                        Toast.makeText(PaymentActivity.this, "Failed to monitor slot status for " + slotId, Toast.LENGTH_SHORT).show();
                    }
                });
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Toast.makeText(PaymentActivity.this, "Failed to monitor Time/" + slotId + "/Trangthai.", Toast.LENGTH_SHORT).show();
        }
    });
  }
}
// Phương thức lấy thông tin thanh toán từ Firebase
```

```
    private void fetchPaymentDetails(String slotId) {
        DatabaseReference slotRef = FirebaseDatabase.getInstance().getReference("Time").child(slotId);

        slotRef.addListenerForSingleValueEvent(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot snapshot) {
                String entryTime = snapshot.child("Entry").getValue(String.class);
                String exitTime = snapshot.child("Exit").getValue(String.class);
                String totalAmount = snapshot.child("Total").getValue(String.class);

                if (entryTime != null && exitTime != null && totalAmount != null) {
                    tvEntryTime.setText("Entry Time: " + entryTime);
                    tvExitTime.setText("Exit Time: " + exitTime);
                    tvTotalAmount.setText("Total Amount: " + totalAmount + " VND");
                    Log.d("PaymentActivity", "Entry Time: " + entryTime);
                    Log.d("PaymentActivity", "Exit Time: " + exitTime);
                    Log.d("PaymentActivity", "Total Amount: " + totalAmount);

                    // Tính tổng tiền sau khi trừ 10,000
                    discountedAmount = Integer.parseInt(totalAmount) - 10000;
                    tvDiscountedAmount.setText("Total Amount Due: " + discountedAmount + " VND");
                } else {
                    Toast.makeText(PaymentActivity.this, "Failed to retrieve payment details.", Toast.LENGTH_SHORT).show();
                }
            }
```

```
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Toast.makeText(PaymentActivity.this, "Failed to retrieve data from Firebase.", Toast.LENGTH_SHORT).show();
        }
    });
}

// Phương thức xử lý thanh toán
private void processPayment() {
    String userId = FirebaseAuth.getInstance().getCurrentUser().getUid();
    DatabaseReference userRef = FirebaseDatabase.getInstance().getReference("users").child(userId);

    userRef.child("balance").addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            Integer currentBalance = snapshot.getValue(Integer.class);

            if (currentBalance != null && currentBalance >= discountedAmount) {
                int newBalance = currentBalance - discountedAmount;

userRef.child("balance").setValue(newBalance).addOnCompleteListener(task -> {
                    if (task.isSuccessful()) {
                        Toast.makeText(PaymentActivity.this, "Payment successful!", Toast.LENGTH_SHORT).show();
                        // Chuyển hướng đến UserBookingsActivity
```

```
                    Intent intent = new Intent(PaymentActivity.this, UserBookingsActivity.class);
                    startActivity(intent);
                    finish();
                } else {
                    Toast.makeText(PaymentActivity.this, "Payment failed. Please try again.", Toast.LENGTH_SHORT).show();
                }
            });
        } else {
            Toast.makeText(PaymentActivity.this, "Insufficient balance.", Toast.LENGTH_SHORT).show();
        }
    }
    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Toast.makeText(PaymentActivity.this, "Failed to retrieve user balance.", Toast.LENGTH_SHORT).show();
    }
  });
 }
}
```

Tệp PaymentActivity.java

```
package com.iot.baidoxethongminh.Activity;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
```

```
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import com.iot.baidoxethongminh.Adapter.UserBookingsActivity;
import com.iot.baidoxethongminh.R;

public class PaymentActivity extends AppCompatActivity {

    private TextView tvSlot, tvEntryTime, tvExitTime, tvTotalAmount,
tvDiscountedAmount;
    private Button btnPay, btnback;
    private int discountedAmount;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_payment);
```

```
    // Ánh xạ các View từ giao diện
    tvSlot = findViewById(R.id.tvSlot);
    tvEntryTime = findViewById(R.id.tvEntryTime);
    tvExitTime = findViewById(R.id.tvExitTime);
    tvTotalAmount = findViewById(R.id.tvTotalAmount);
    tvDiscountedAmount = findViewById(R.id.tvDiscountedAmount);
    btnPay = findViewById(R.id.btnPay);
    btnback = findViewById(R.id.btnback);

    // Khởi tạo và bắt đầu theo dõi trạng thái của các slot
    monitorFirebaseForSlotChanges();

    btnback.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            startActivity(new Intent(PaymentActivity.this, MainActivity.class));
        }
    });

    // Thiết lập sự kiện nhấn nút "Pay"
    btnPay.setOnClickListener(view -> processPayment());
}

// Phương thức theo dõi thay đổi trạng thái của các slot trên Firebase
private void monitorFirebaseForSlotChanges() {
    for (int i = 1; i <= 4; i++) {
        String slotId = "Slot" + i;
```

```
        DatabaseReference timeSlotRef = FirebaseDatabase.getInstance().getReference("Time").child(slotId).child("Trangthai");
        DatabaseReference slotStatusRef = FirebaseDatabase.getInstance().getReference("Status").child(slotId);

        timeSlotRef.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(@NonNull DataSnapshot timeSnapshot) {
                Integer trangthai = timeSnapshot.getValue(Integer.class);

                if (trangthai != null && trangthai == 0) {
                    slotStatusRef.addValueEventListener(new ValueEventListener() {
                        @Override
                        public void onDataChange(@NonNull DataSnapshot statusSnapshot) {
                            String status = statusSnapshot.getValue(String.class);

                            if ("Ok".equals(status)) {
                                tvSlot.setText("Slot: " + slotId);
                                fetchPaymentDetails(slotId);
                            }
                        }

                        @Override
                        public void onCancelled(@NonNull DatabaseError error) {
                            Toast.makeText(PaymentActivity.this, "Failed to monitor slot status for " + slotId, Toast.LENGTH_SHORT).show();
                        }
```

```
                });
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Toast.makeText(PaymentActivity.this, "Failed to monitor Time/" +
slotId + "/Trangthai.", Toast.LENGTH_SHORT).show();
        }
    });
  }
}
// Phương thức lấy thông tin thanh toán từ Firebase
private void fetchPaymentDetails(String slotId) {
  DatabaseReference slotRef =
FirebaseDatabase.getInstance().getReference("Time").child(slotId);

  slotRef.addListenerForSingleValueEvent(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
      String entryTime = snapshot.child("Entry").getValue(String.class);
      String exitTime = snapshot.child("Exit").getValue(String.class);
      String totalAmount = snapshot.child("Total").getValue(String.class);

      if (entryTime != null && exitTime != null && totalAmount != null) {
        tvEntryTime.setText("Entry Time: " + entryTime);
        tvExitTime.setText("Exit Time: " + exitTime);
        tvTotalAmount.setText("Total Amount: " + totalAmount + " VND");
        Log.d("PaymentActivity", "Entry Time: " + entryTime);
        Log.d("PaymentActivity", "Exit Time: " + exitTime);
```

```
            Log.d("PaymentActivity", "Total Amount: " + totalAmount);

            // Tính tổng tiền sau khi trừ 10,000
            discountedAmount = Integer.parseInt(totalAmount) - 10000;
            tvDiscountedAmount.setText("Total Amount Due: " +
discountedAmount + " VND");
        } else {
            Toast.makeText(PaymentActivity.this, "Failed to retrieve payment
details.", Toast.LENGTH_SHORT).show();
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {
        Toast.makeText(PaymentActivity.this, "Failed to retrieve data from
Firebase.", Toast.LENGTH_SHORT).show();
    }
  });
 }
 // Phương thức xử lý thanh toán
 private void processPayment() {
  String userId = FirebaseAuth.getInstance().getCurrentUser().getUid();
  DatabaseReference userRef =
FirebaseDatabase.getInstance().getReference("users").child(userId);

  userRef.child("balance").addListenerForSingleValueEvent(new
ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot) {
```

```
            Integer currentBalance = snapshot.getValue(Integer.class);

            if (currentBalance != null && currentBalance >= discountedAmount) {
                int newBalance = currentBalance - discountedAmount;

userRef.child("balance").setValue(newBalance).addOnCompleteListener(task -> {
                    if (task.isSuccessful()) {
                        Toast.makeText(PaymentActivity.this, "Payment successful!",
Toast.LENGTH_SHORT).show();
                        // Chuyển hướng đến UserBookingsActivity
                        Intent intent = new Intent(PaymentActivity.this,
UserBookingsActivity.class);
                        startActivity(intent);
                        finish();
                    } else {
                        Toast.makeText(PaymentActivity.this, "Payment failed. Please
try again.", Toast.LENGTH_SHORT).show();
                    }
                });
            } else {
                Toast.makeText(PaymentActivity.this, "Insufficient balance.",
Toast.LENGTH_SHORT).show();
            }
        }
        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Toast.makeText(PaymentActivity.this, "Failed to retrieve user balance.",
Toast.LENGTH_SHORT).show();
        }
```

```
        });
    }
}
```

Tệp Notifications.java

```
package com.iot.baidoxethongminh.Adapter;

import android.app.Activity;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.content.Context;
import android.content.pm.PackageManager;
import android.os.Build;

import androidx.core.app.ActivityCompat;
import androidx.core.app.NotificationCompat;
import androidx.core.content.ContextCompat;

import com.google.firebase.database.DataSnapshot;
import com.google.firebase.database.DatabaseError;
import com.google.firebase.database.DatabaseReference;
import com.google.firebase.database.FirebaseDatabase;
import com.google.firebase.database.ValueEventListener;
import android.Manifest;

public class Notifications {
    private Context context;
    private NotificationManager notificationManager;
    private DatabaseReference databaseReference;
```

```
    // Constructor
    public Notifications(Context context) {
        this.context = context;
        this.notificationManager = (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
        this.databaseReference = FirebaseDatabase.getInstance().getReference();
        // Kiểm tra và yêu cầu quyền thông báo nếu cần thiết
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            if (ContextCompat.checkSelfPermission(context, Manifest.permission.POST_NOTIFICATIONS) != PackageManager.PERMISSION_GRANTED) {
                ActivityCompat.requestPermissions((Activity) context, new String[]{Manifest.permission.POST_NOTIFICATIONS}, 101);
            }
        }
        createNotificationChannel();
        monitorSlotChanges();
    }
    // Tạo kênh thông báo (chỉ cần thiết với Android O và cao hơn)
    private void createNotificationChannel() {
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            String channelId = "PARKING_SLOT_NOTIFICATION_CHANNEL";
            String channelName = "Parking Slot Notifications";
            NotificationChannel channel = new NotificationChannel(channelId, channelName, NotificationManager.IMPORTANCE_HIGH);
            notificationManager.createNotificationChannel(channel);
        }
    }
```

```
    // Giám sát các thay đổi từ Firebase cho tất cả các Slot
    private void monitorSlotChanges() {
        databaseReference.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                checkAndNotifyUser(dataSnapshot, "Slot1");
                checkAndNotifyUser(dataSnapshot, "Slot2");
                checkAndNotifyUser(dataSnapshot, "Slot3");
                checkAndNotifyUser(dataSnapshot, "Slot4");
            }
            @Override
            public void onCancelled(DatabaseError databaseError) {
                // Xử lý lỗi nếu cần
            }
        });
    }
    // Kiểm tra và gửi thông báo nếu điều kiện được đáp ứng
    private void checkAndNotifyUser(DataSnapshot dataSnapshot, String slot) {
        String statusSlot =
dataSnapshot.child("Status").child(slot).getValue(String.class);
        Long trangThaiSlot =
dataSnapshot.child("Time").child(slot).child("Trangthai").getValue(Long.class);

        if ("No".equals(statusSlot) && trangThaiSlot != null && trangThaiSlot == 0)
{
            // Gửi thông báo
            sendNotification("Bạn đã đậu đúng vị trí " + slot);
        } else if (trangThaiSlot != null && trangThaiSlot == 0) {
            sendNotification("Vui lòng đậu vào vị trí " + slot);
```

```
        }
    }

    // Gửi thông báo đến người dùng
    private void sendNotification(String message) {
        String channelId = "PARKING_SLOT_NOTIFICATION_CHANNEL";
        NotificationCompat.Builder builder = new
NotificationCompat.Builder(context, channelId)
                .setSmallIcon(android.R.drawable.ic_dialog_info)
                .setContentTitle("Thông báo đậu xe")
                .setContentText(message)
                .setPriority(NotificationCompat.PRIORITY_HIGH)
                .setAutoCancel(true);

        notificationManager.notify((int) System.currentTimeMillis(), builder.build());
    }
}
```