PROJECT REPORT

DELIVERABLES

A Smart Delivery Notification System for Building Complexes

TEAM MEMBERS

ROBERT STEFAN GONTARU
STANISLAV USTINOV
ERNEST PATRYK WARZUCHOWSKI
OWEN FRANK O'CONNOR
THOMAS JACK WALSH
EMMANUELLA KUNMI AYODEJI

as of 28th April 2025

Contents

1

INTRODUCTION

Deliverables is designed to streamline the parcel handling process in building complexes. Instead of relying on manual notifications by reception staff, our solution automates the entire process. When a parcel arrives at the concierge desk, a simple photo of the delivery label is taken. Advanced AI vision and image processing extract the recipient's name and flat number. Instantly, a personalized message is sent via WhatsApp directly to the resident. This automation not only speeds up communication but also enhances the overall resident experience by ensuring timely, accurate notifications.

2

REQUIREMENTS

2.1 FUNCTIONAL REQUIREMENTS

- Residents must be able to register by uploading a contract via the WhatsApp bot, enabling seamless onboarding.
- Admins must have the ability to manually create, update, and delete packages, ensuring flexibility in package management.
- The system must track the status of each package (e.g., collected or not collected) and provide real-time updates to users.
- Automated WhatsApp notifications must be sent to residents when a parcel arrives, ensuring timely communication.
- The system must use AI-powered image processing to accurately extract recipient details (e.g., name and flat number) from parcel labels, with a fallback mechanism for manual input in case of errors.
- Residents must confirm their phone numbers during registration to ensure accurate delivery notifications.
- An administrative dashboard must allow monitoring of system performance, reviewing notifications, and managing configurations.
- Residents must be able to update their registration details after onboarding, ensuring flexibility in maintaining accurate information.

2.2 NON-FUNCTIONAL REQUIREMENTS

- **Security:** The app must feature user authentication, role-based access control, and secure data storage to protect sensitive resident information.
- **Role-Based Access Control (RBAC):** RBAC is enforced to restrict user access based on their role, minimizing unauthorized access and supporting compliance.
- **Scalability:** The app must be scalable to support hundreds of thousands of residents across thousands of complexes internationally, leveraging Microsoft Azure for backend infrastructure.

- **Availability:** The app must be available 24/7/365, with minimal downtime to ensure uninterrupted service.
- **Usability:** The software must be intuitive, user-friendly, and easy to use, with a sleek interface designed for concierges and residents.
- **Performance:** The system must handle high volumes of concurrent requests efficiently, ensuring a smooth user experience.
- **Maintainability:** The system must be easy to maintain, with modular components such as the image processing module, WhatsApp bot, and backend APIs. CI/CD pipelines must support rollback in case of deployment failures.
- **Accuracy:** AI-driven image processing must minimize errors in extracting recipient details from parcel labels.

2.3 SYSTEM REQUIREMENTS

- Sensitive data must be handled with care, ensuring proper access control and data privacy.
- The system must integrate with Azure AI for image processing to extract recipient details from parcel labels.
- The backend must be built using Microsoft Azure to ensure scalability, security, and reliability.
- The WhatsApp bot must be integrated with the backend system to send personalized delivery notifications in real-time.
- A secure database must be used to store and manage resident information, ensuring data integrity and accessibility.
- The administrative dashboard must provide tools for monitoring system performance, reviewing notifications, and managing configurations.
- The system must support cloud deployment with CI/CD pipelines for seamless updates and maintenance, including rollback capabilities.

3

DESIGN

3.1 USE CASE DIAGRAM

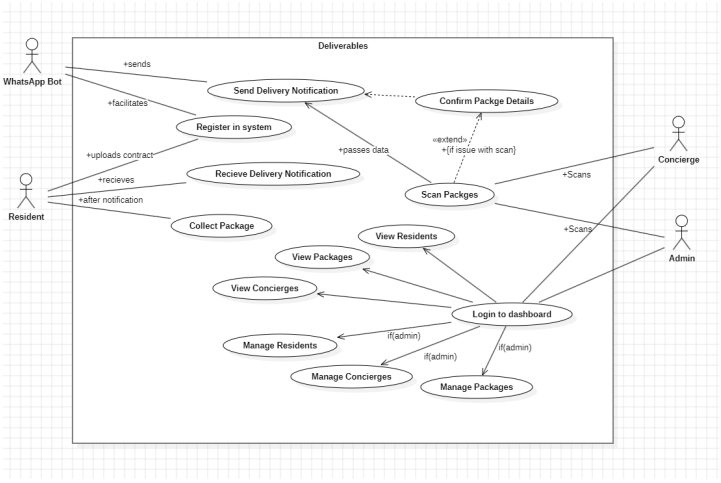


Figure 1: Use Case Diagram

3.2 CLASS DIAGRAM

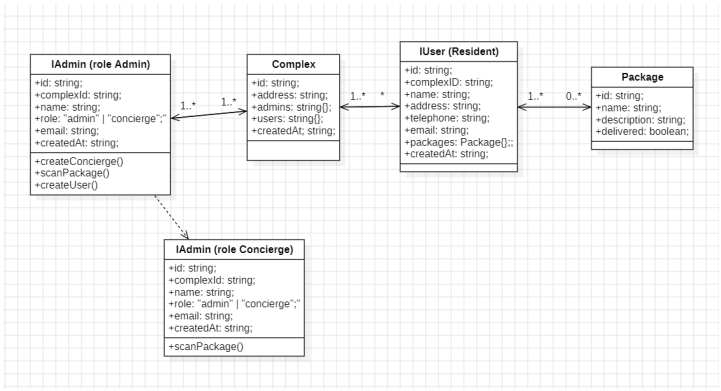


Figure 2: Class Diagram

3.3 ARCHITECTURAL DIAGRAMS

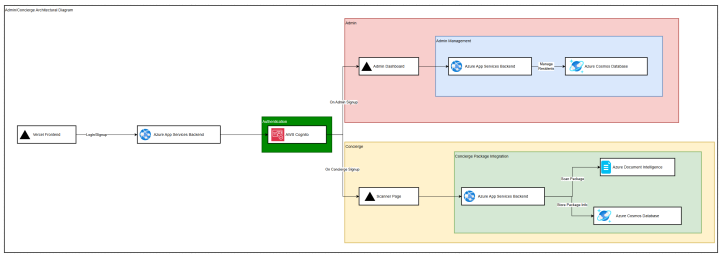


Figure 3: Admin Architecture Diagram

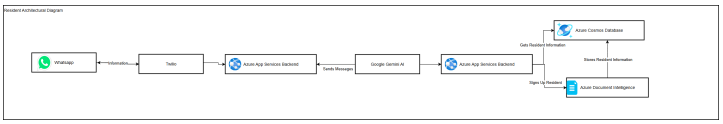


Figure 4: Resident Architecture Diagram

Figure 5: WhatsApp Bot via Twilio Diagram

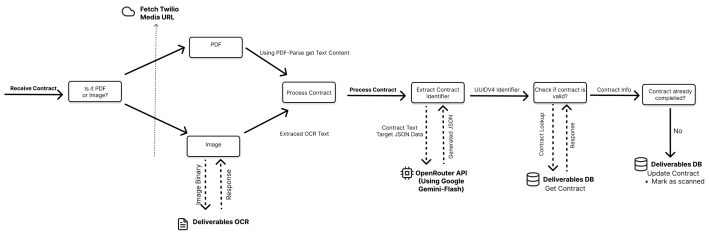


Figure 6: Resident Onboarding via Contract Diagram

4

SOLUTION

4.1 DATABASE DESIGN

The database is structured to efficiently handle resident information, package delivery details, and notification logs. A NoSQL solution proved optimal, allowing intuitive partitioning by complex without cross-complex queries. This improved performance for specific data subsets and supported horizontal scaling, where each server independently manages a few complexes.

Real-time updates were not critical; instead, a serverless architecture was prioritized to significantly reduce compute costs and maintain budget constraints.

Azure Cosmos DB was selected for its high-performance operations, particularly critical lookups such as mapping resident names to user IDs.

A reference-based data model was chosen over a nested structure after considering real-world cases, like residents living in multiple complexes. This minimizes data inconsistency risks and keeps the system flexible and easier to maintain.

The database includes the following collections:

- **Admin**: Administrator details, including associated complexes, name, role (admin or concierge), email address, and account creation date.
- **Complex**: Complex information, including address, assigned concierges, admins, associated residents, and creation date.
- **User**: Resident details such as name, unit number, phone number, and email address.
- **Package**: Package tracking information, including delivery dates, content description, and intended recipient.
- **Contracts**: Contract agreements signed by users through the WhatsApp bot for onboarding and validation.

The **Complex** collection remains central to data organization.

Enhanced OCR and Information Extraction

To streamline concierge workflows, Azure Document Intelligence was used for OCR, as its line-by-line reading suited structured package

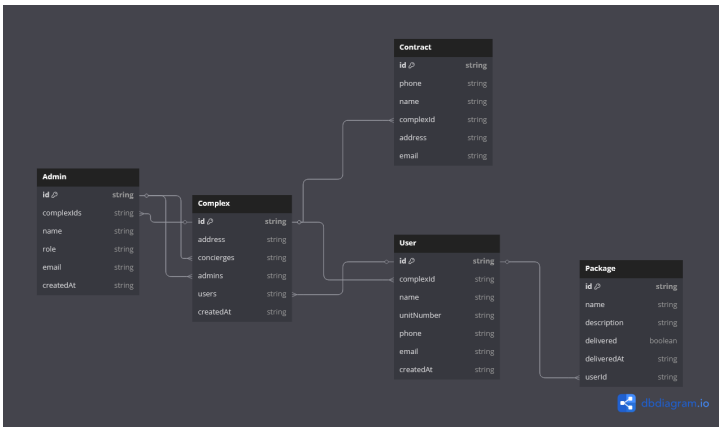


Figure 7: Database schema overview

labels. Regex-based extraction was initially attempted but proved unreliable due to label format variance and difficulty distinguishing recipients from senders.

Address validation was explored but found too slow and costly. Instead, extracted text lines were fed into a fine-tuned LLM, with Google Gemini Flash 2 offering the best balance of speed, cost, and accuracy.

For security, only raw text (no metadata) was provided to the model, which outputs data conforming to a strict TypeScript interface with additional validation.

This improved extraction reliability and enables future features like generating personalized messages based on package content sentiment.

The codebase is structured as a monorepo managed with Lerna for seamless development and deployment.

4.2 KEY FEATURES AND FUNCTIONALITY

The solution offers the following key features:

- **Smart Scanning**: Uses Azure AI to extract recipient details from parcel labels.
- **Automated Notifications**: Sends personalized WhatsApp messages upon parcel arrival.
- **Resident Registration**: Allows residents to securely register and verify information.
- **Administrative Dashboard**: Provides tools for monitoring system performance and managing notifications.
- **Google Maps API Integration**: Boosts dashboard interactivity and context.

These features address the problems of current package management solutions by automating manual tasks and improving communication efficiency.

4.3 IMPLEMENTATION DETAILS

The solution is built using TypeScript and a modern tech stack:

- **Frontend**: React and Next.js for a responsive UI.
- **Backend**: Node.js and Express, hosted on Microsoft Azure for scalability.
- **Database**: CosmosDB for secure, efficient data storage.
- **AI Integration**: Azure AI for image processing and document analysis.
- **Messaging**: WhatsApp bot for real-time notifications.
- **Authentication**: AWS Cognito manages login securely, ensuring GDPR compliance and resident data protection.

4.4 CHALLENGES AND SOLUTIONS

- **Accurate Image Processing**: Fine-tuned Azure AI models improved recipient detail extraction.
- **Data Security**: Secure authentication and encryption protect resident registration and database access.
- **Scalability**: Hosting on Azure enables easy scaling of backend resources as needed.

4.5 CONCLUSION

The Smart Delivery Notification System effectively automates the parcel notification process, reducing manual workload and enhancing the resident experience. Future improvements could include multi-language notification support and integration with additional messaging platforms.

5 USER INTERFACE

5.1 DESIGN PRINCIPLES

The UI design for this project adheres to the following principles:

- **Simplicity:** The interface is designed to be intuitive and easy to navigate.
- **Consistency:** Consistent design patterns and visual elements are used across all pages.
- **Responsiveness:** The scanner's UI is optimized for both desktop and mobile devices.

5.2 DESKTOP INTERFACE

5.2.1 *Landing Page*

The desktop landing page serves as the entry point for users. It provides quick access to login/sign-up features and advertises the main functions of the app.

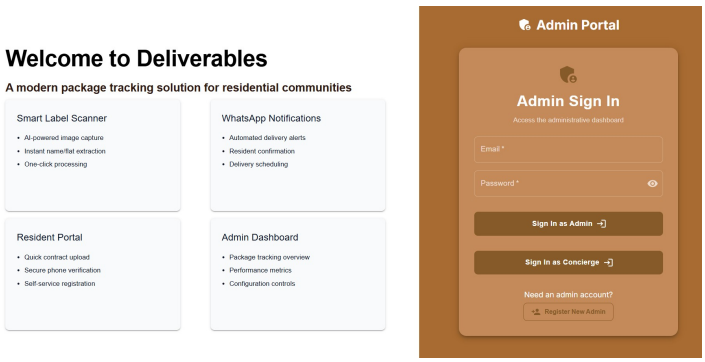


Figure 8: Landing Page

5.2.2 *Dashboard Overview*

The dashboard provides an overview of the application's key metrics and functionalities.

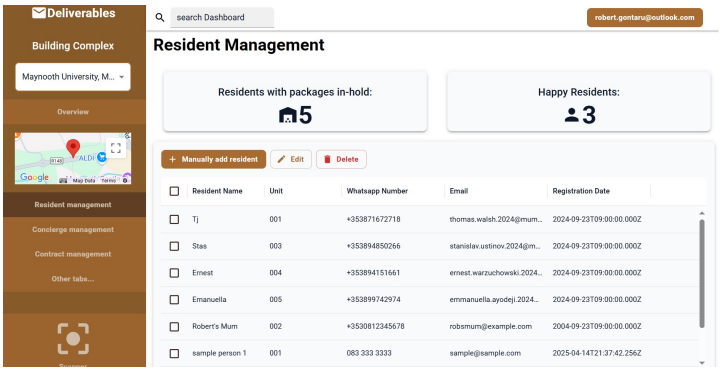


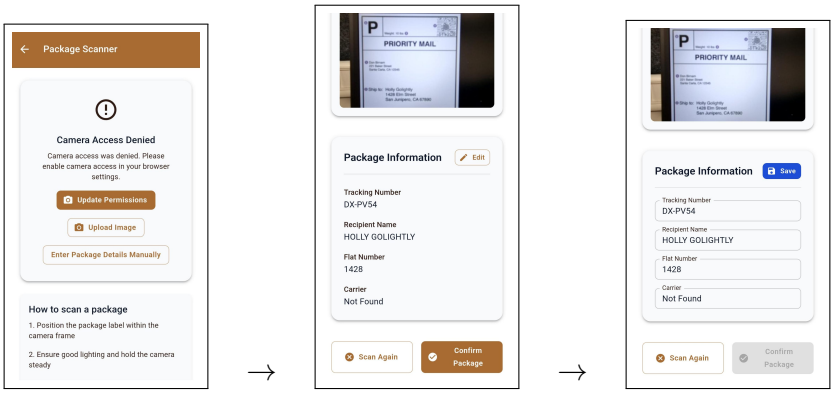
Figure 9: Dashboard

Description: The dashboard includes real-time summary statistics, a sidebar for easy-navigation and detailed lists of relevant information.

5.3 MOBILE INTERFACE

5.3.1 *Mobile Scanning Page*

The mobile scanning page is designed for users to scan items or documents using their mobile devices.



Description: The mobile scanning page includes the following elements:

- **Scan Button:** Allows users to initiate the scanning process.
- **Preview Area:** Displays a preview of the scanned label for verification.
- **Edit Button:** Enables users to manually edit extracted details if the OCR output is incorrect.
- **Upload Option:** Provides an alternative to upload images directly from the device.

The design ensures that scanning is quick and efficient, even on smaller screens.

5.4 FUTURE ENHANCEMENTS

Planned improvements to the UI include:

- Adding a logs page to provide admins with a detailed history of system activities and events.
- Implementing a user settings page to allow users to customize their preferences, such as notification settings and account details.
- Redesigning the dashboard including better data visualization.
- Adding dark mode support for improved accessibility.

5.5 CONCLUSION

The user interface is designed to provide a seamless and enjoyable experience for admins. By employing good design principles and focusing on simplicity, consistency and helpful feedback, the project aims to deliver a UI that meets the needs of its target audience.

6

REFERENCES

- Microsoft Azure Documentation: https://docs.microsoft.com/azure/
- Azure Cosmos DB Documentation: https://docs.microsoft.com/azure/cosmos-db/
- Azure AI Document Intelligence: https://learn.microsoft.com/azure/ai-services/document-intelligence/
- Twilio Documentation: https://www.twilio.com/docs
- AWS Cognito Documentation: https://docs.aws.amazon.com/cognito/
- React Documentation: https://react.dev/
- Next.js Documentation: https://nextjs.org/docs/
- Node.js Documentation: https://nodejs.org/en/docs/
- Express Documentation: https://expressjs.com/
- Lerna Documentation: https://lerna.js.org/
- GitHub Repository: https://github.com/robi2711/Microsoft-Project-Deliverables