# CS 5520 – Mobile Application Development
# Final Project: Driver's App for Logistics System

Student: Sukhrobbek Ilyosbekov
Date: 12/04/2025

## Introduction

The Logistics Driver App is a cross-platform mobile application designed for truck drivers to manage their daily operations, including viewing assigned loads, tracking deliveries, monitoring performance statistics, and managing their profiles. The application was developed as a complete migration from a legacy .NET MAUI application to a modern **Kotlin Multiplatform (KMP)** solution with **Compose Multiplatform** for the UI layer. This project demonstrates the power of code sharing between Android and iOS platforms, achieving approximately 90% shared code while maintaining native performance and platform-specific integrations.

**Features:**

- Authentication: User Login with ROPC flow, session management using the DataStore and auto-login with token refresh technique
- Dashboard: Display assigned truck details and driver. Real-time list of currently assigned loads.
- Load Management: Displays load details with comprehensive view of pickup and delivery information. Google Maps integration for navigation. Origin and destination with formatted addresses
- Statistics & Analytics: Displays weekly/monthly stats for gross earnings, driver share, and distance metrics. Custom-built Bar and Line charts with animations to display driver income.
- Past Loads: View completed loads from the past 90 days
- Account Management: View and update user details. Custom phone input field for international phone number format with country codes
- Settings: Toggle between Miles and Kilometers. Change app language.
- Real-time tracking: Foreground service for continuous GPS updates and reporting driver's location to backend on every 30 seconds. Proximity detection when driver is near pickup/delivery locations.
- Adaptive UI: supports both portrait and landscape modes.
- Clean, modular codebase with reusable UI components.

**SDK Information:**

- Target SDK: 36 (Android 16)
- Minimum SDK: 28 (Android 9)
- Compile SDK: 36

**Devices Tested:** Pixel 9 Pro API 36 (AVD) - Android 16**,** Samsung Galaxy S25 Ultra - Android 16

## Changes from legacy MAUI Application

The new driver app is rewritten completely from scratch in Kotlin KMP. The legacy app was written in .NET MAUI framework which was in C# and XAML languages. The main reason for the migration is MAUI's instability and the lengthy project update process after a new .NET version is released. MAUI lags behind the native Android and iOS SDKs. The new Android SDK will not be available in MAUI immediately after the new version is released. The Syncfusion UI library has become a paid library; there was a free community license, but new versions require payment and a commercial license.

Here is list of changes compared to the legacy application:

- Architecture Changes:

| Aspect | Legacy MAUI | New KMP |
|---|---|---|
| Language | C# | Kotlin |
| Platform Sharing | .NET Standard libraries | Kotlin Multiplatform (commonMain) |
| Dependency Injection | Microsoft.Extensions.DI | Koin (Multiplatform) |
| Serialization | System.Text.Json | Kotlinx Serialization |
| Navigation | Shell Navigation | Jetbrains Navigation Compose |

- UI Framework:

| Aspect | Legacy MAUI | New KMP |
|---|---|---|
| UI Paradigm | XAML based declarative language | Kotlin Compose declarative API |
| UI Library | Syncfusion UI components (not free) | Material3 (free) |
| Charts | Syncfusion Charts | Custom Canvas based charts |
| Theming | MAUI Styles/Resources | Compose MaterialTheme |
| State Management | MVVM with INotifyPropertyChanged | StateFlow + collectAsState |

## Key Improvements

List of key improvements over the legacy app that is missing these features

- New authentication method: instead of using OIDC browser-based authentication, implemented ROPC based authentication which is directly typing email and password in the login screen and signing in with authentication backend.
- Used custom charts: the legacy app had third-party charts from Syncfusion and requires license. Built custom charts based on Canvas and multi-cross platform compatible. It supports animation and touch interaction for data point selection. The new charts code is in the commonMain/kotlin/ui/components/charts folder.
- Improved location services: Implemented native foreground service with FusedLocationProvider to report accurate location to the backend. Proximity detection algorithm for pickup/delivery confirmation. Implemented SignalR real-time communication with the backend service.
- Automatic API client code generation: In the old application, I had to constantly update the client code (http client, DTO models, etc.) every time the backend API schema changed. In the new application, I added the OpenAPI generator plugin for generating type-safe API clients and DTO models based on the api-spec.json file, which significantly saves time.
- New dashboard and statistics screens: implemented a new dashboard and statistics screens with Material3 components.
- New layout: The old app used a sidebar for navigation, which was inconvenient. In the new app, I implemented a new layout with navigation in the bottom bar.
- New settings screen: added distance measure unit and switch language options.

## Concepts applied from the class materials

Here is list of concepts applied to the new project from the class:

- Android lifecycles: The **LocationTrackingService** (androidMain/kotlin/service/LocationTrackingService.kt file) extends Android Service with proper **onCreate()**, **onStartCommand()**, **onDestroy()** lifecycle handling.
- Kotlin fundamentals: extensively used Kotlin specific language features such as nullable types, extension functions, higher-order functions, lambda functions, Slot API and OOD principles.
- Jetpack compose and view models: all UI is written in Jetpack Compose and used MVVM architecture to separate UI and business logic. UI and ViewModel files are in the commonMain/kotlin/ui and commonMain/kotlin/viewmodel folders.

- Layout Design: used Modifiers such as **.fillMaxWidth()**, **.padding()**, **.clickable()**, **.height()**, **.weight()**, For all screens used **Row** and **Column** components to build layouts. (Screen files are in the commonMain/kotlin/ui/screens folder)
- Advanced UI Components: used coroutines **viewModelScope.launch {}** for async API calls and **CoroutineScope(Dispatchers.IO)** in services (commonMain/kotlin/viewmodel and commonMain/kotlin/service folders) **LazyColumn** with contentPadding and verticalArrangement in all list screens. **LazyColumn** with **items()** for efficient rendering of loads list. Text measurement with **rememberTextMeasurer()** for chart label positioning for intrinsic sizing.
- Sensors & GPS: Implemented GPS access with **FusedLocationProviderClient** for high-accuracy location updates (androidMain/kotlin/service/LocationTrackingService.kt). Implemented runtime permission requesting (androidMain/kotlin/permission folder)
- Firebase services: Implemented firebase cloud messaging for push notification in the DriverFirebaseMessagingService Android service (androidMain/kotlin/service/DriverFirebaseMessagingService.kt)

## Objectives

**Primary**

- Rewrite MAUI app to Kotlin ✔
- Using Material3 components instead of Syncfusion from MAUI app ✔
- Login screen ✔
- Authentication: Login with backend ✔
- Update load status ✔
- User data screen
- Build APIs and connect with the backend ✔

**Secondary**

- Driver statistics with charts ✔
- Push notification ✔
- Real-time location tracking and reporting to backend ✔
- Add google map to the main screen ✔
- Shared Compose UI instead of Android only ✔

**Tertiary**

- Support for trips API ❌
- Biometric authentication ❌
- Direct login without redirecting to Identity website ✔
- iOS implementation ⚠
- Upload load invoice after delivering ❌
- ~~Native Mapbox map layer instead of Google Map's HTML iframe~~
- Chat with dispatcher within app ❌
- Support for multiple languages ⚠

**Legends**:

- ✔ - Completely Implemented
- ⚠ - Partially Implemented
- ❌ - Not Implemented yet
- ~~Strikethrough~~ – Cancelled feature or replaced with another feature.

## Project Reflection

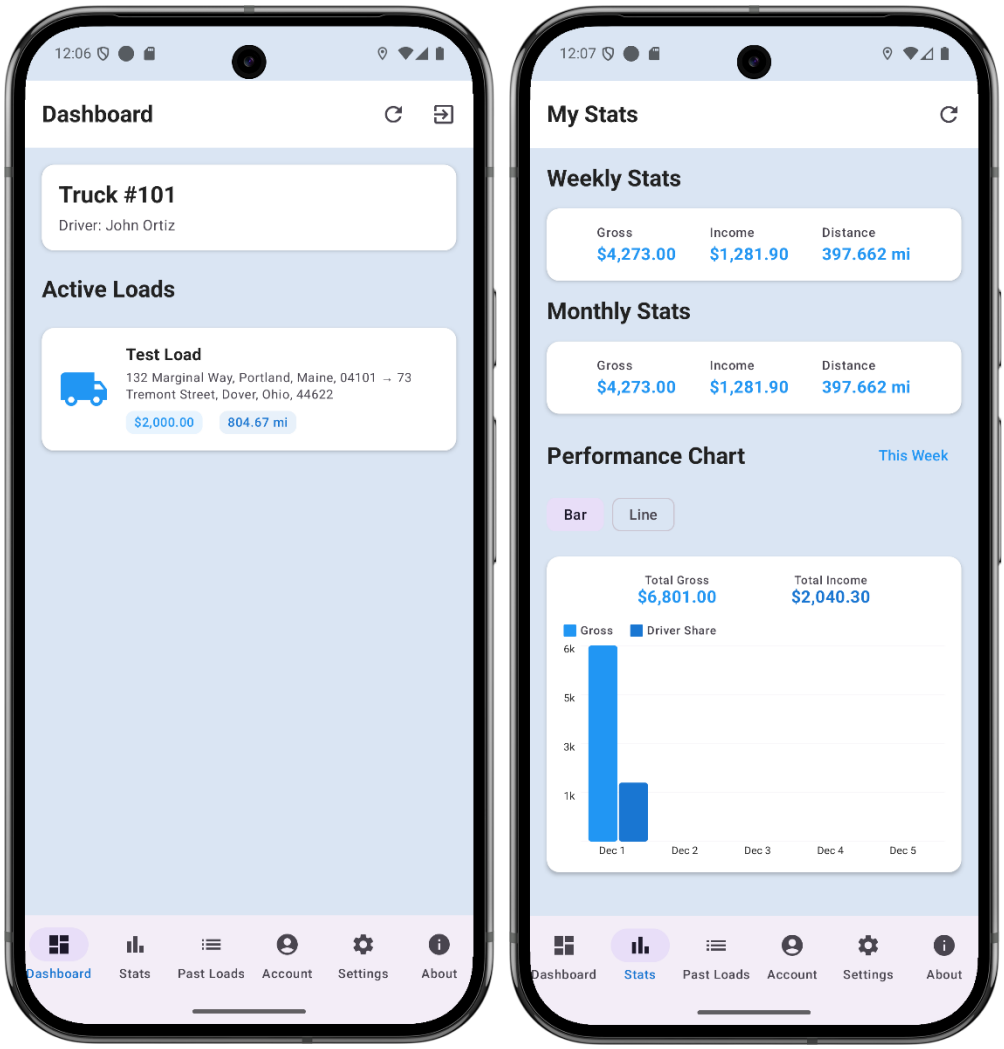
Working on this project, I gained hands-on experience developing multiplatform apps with Kotlin (KMP). KMP is a relatively new framework compared to established frameworks like Flutter and React Native, but it has great potential because it allows us to share 90% of our code between Android and iOS, saving us significant time in the future if we decide to build an iOS version of the app. Traditionally, Kotlin was known as an Android-only language, but over the past few years, JetBrains has been actively investing in KMP and Compose Multiplatform to make them functional and powerful. Now, with Kotlin, we can write apps for iOS, desktop, and web platforms. Beside class concepts, I learned advanced Kotlin features such as expect/actual pattern, infix functions, Canvas composable component, dependency injection with Koin library, building cross-platform http client with Ktor library, differences between Android only Compose and cross-platform Compose API, difference and similarities between Kotlin and C# languages, advanced Material3 components, JetBrains navigation system with multiple screens, and implementing background service in native Android which was tricky and has limitations.

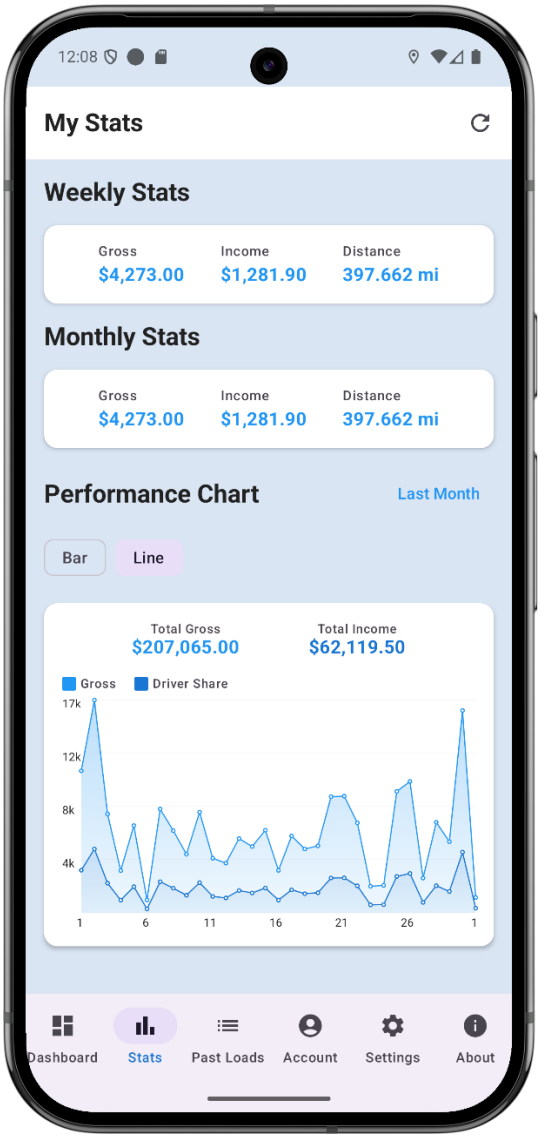
## Semester Reflection

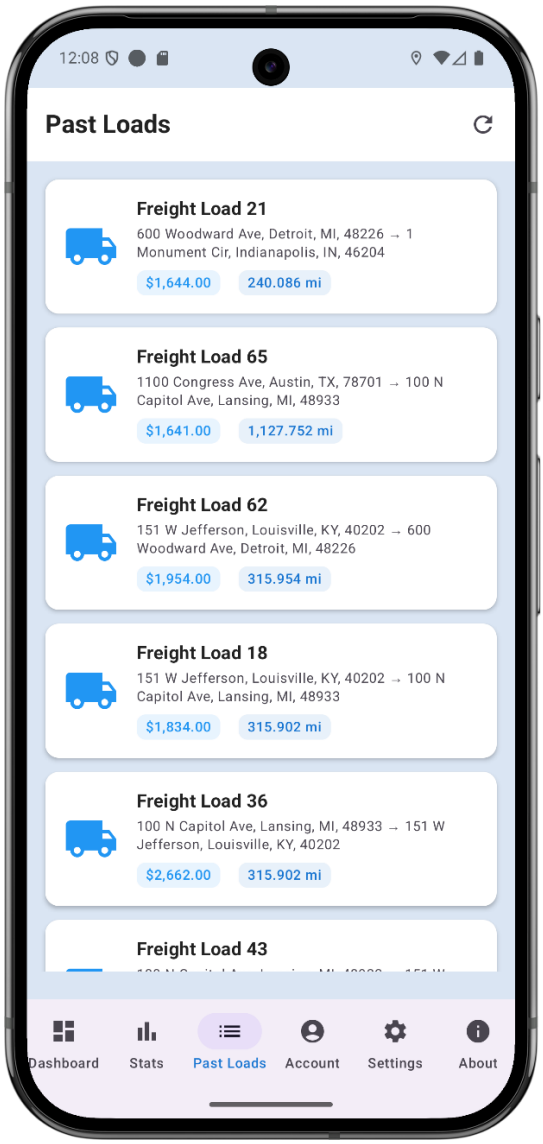
Throughout this semester, I developed a comprehensive skill set in modern mobile development: I mastered Kotlin not just at a basic level, but at an advanced level in just a couple of months and thoroughly explored the Compose API. Before this semester, I already knew how to develop mobile apps using other frameworks, such as React Native and MAUI, but this was my first experience developing mobile apps natively in Kotlin, rather

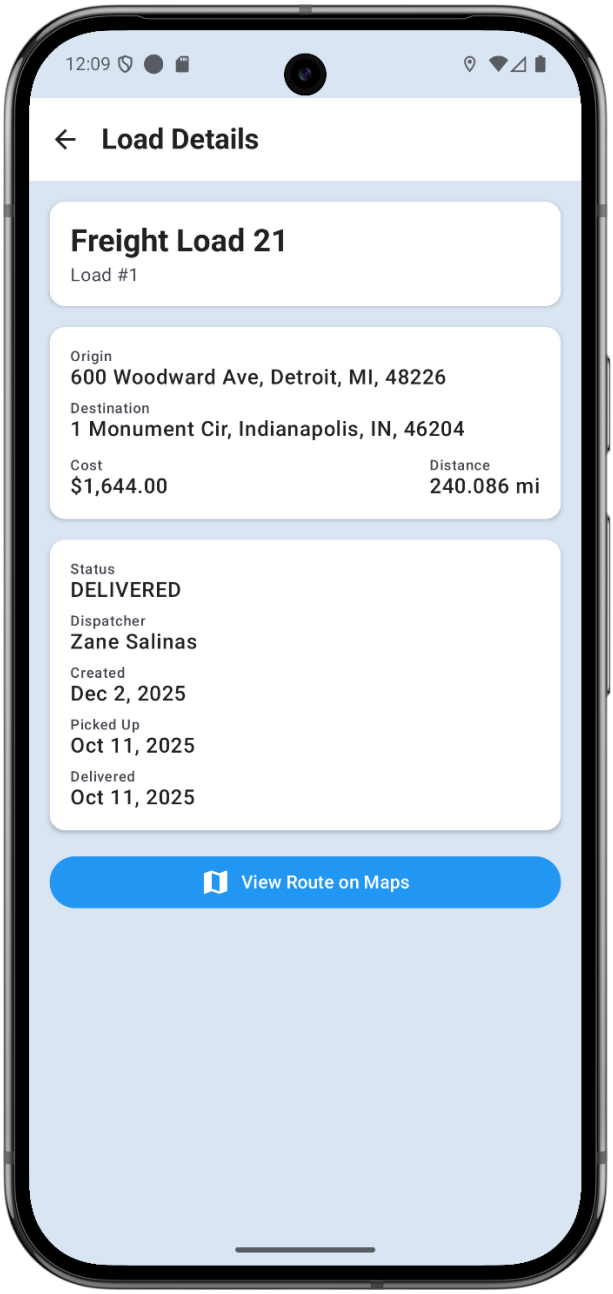
than using wrapper frameworks. This experience helped me quickly master Kotlin and Compose. From now on, I will exclusively use native Kotlin for mobile app development, rather than wrapper frameworks like React Native and MAUI, as they have the challenge of constantly updating to new versions.

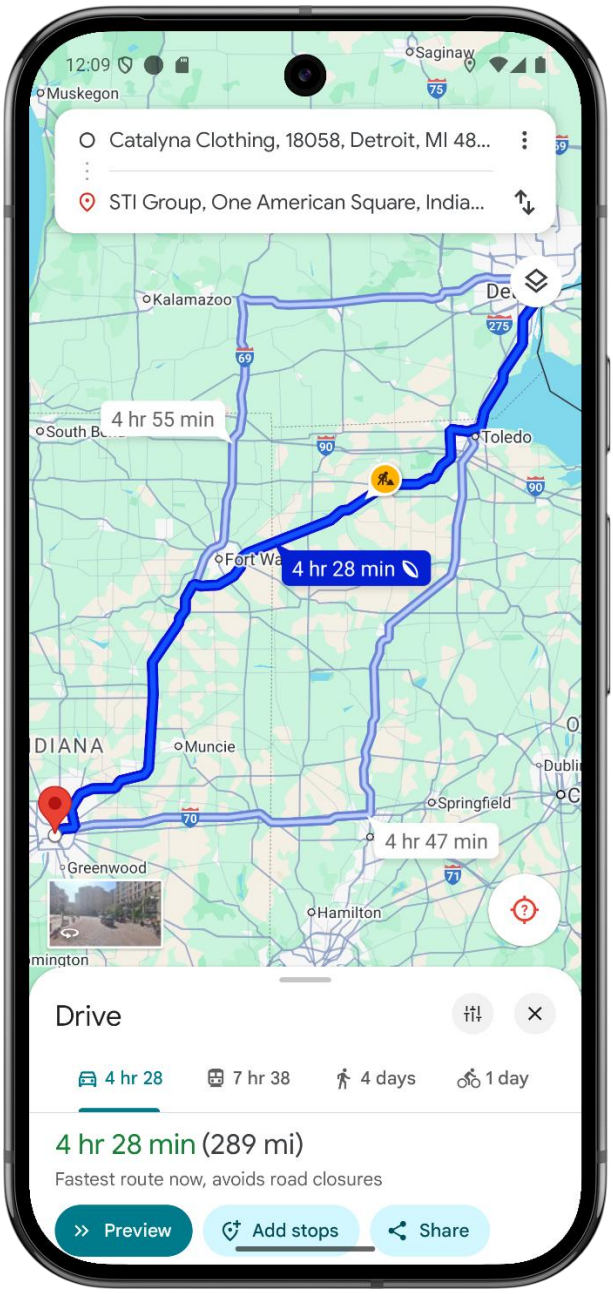
# Screenshots

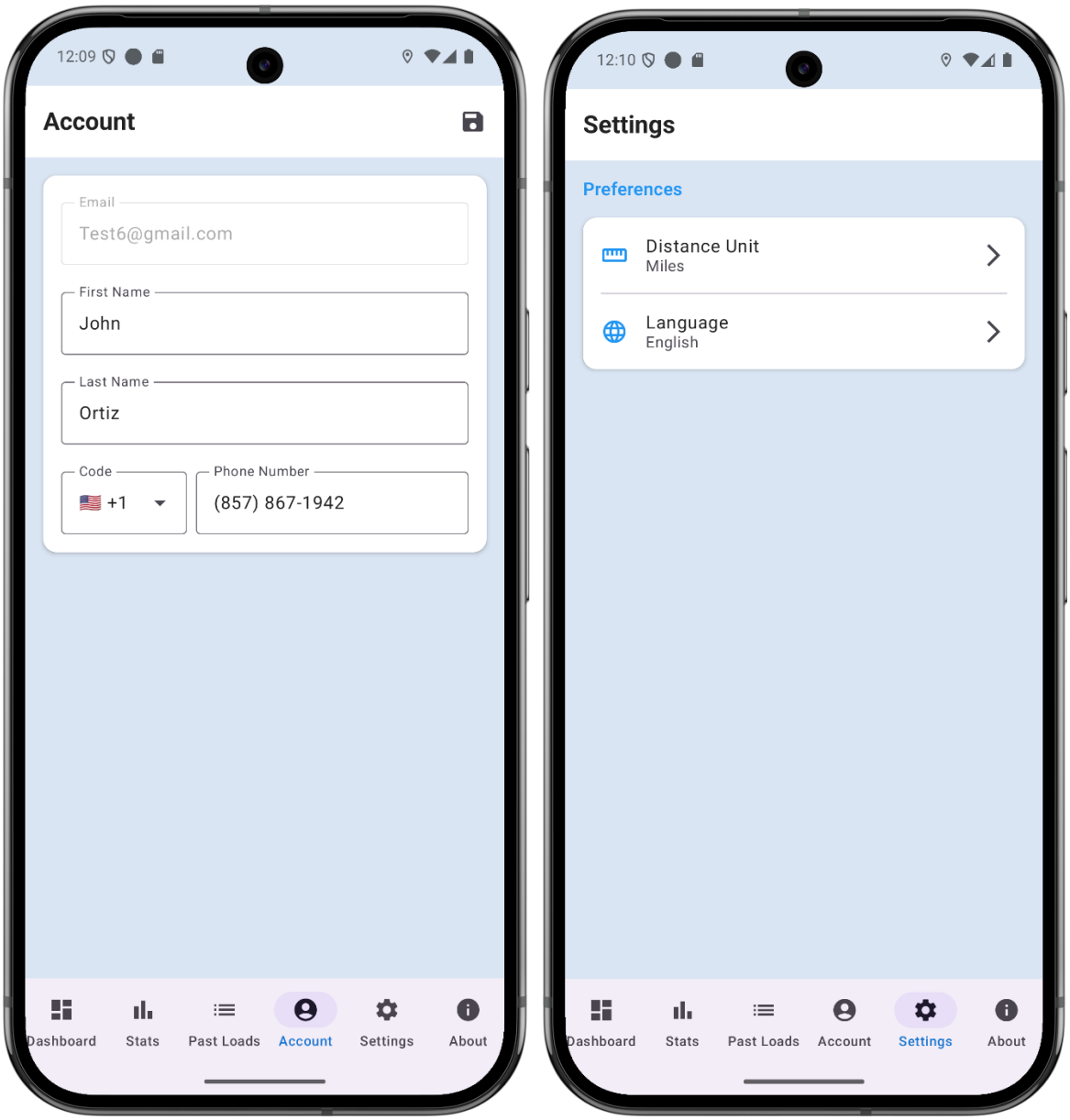



12:08
My Stats
Weekly Stats
Gross
$4,273.00
Income
$1,281.90
Distance
397.662 mi
Monthly Stats
Gross
$4,273.00
Income
$1,281.90
Distance
397.662 mi
Performance Chart
Last Month
Bar
Line
Total Gross
$207,065.00
Total Income
$62,119.50
Gross
Driver Share
17k
12k
8k
4k
1
6
11
16
21
26
1
Dashboard
Stats
Past Loads
Account
Settings
About


12:08
Past Loads
Freight Load 21
600 Woodward Ave, Detroit, MI, 48226 → 1 Monument Cir, Indianapolis, IN, 46204
$1,644.00
240.086 mi
Freight Load 65
1100 Congress Ave, Austin, TX, 78701 → 100 N Capitol Ave, Lansing, MI, 48933
$1,641.00
1,127.752 mi
Freight Load 62
151 W Jefferson, Louisville, KY, 40202 → 600 Woodward Ave, Detroit, MI, 48226
$1,954.00
315.954 mi
Freight Load 18
151 W Jefferson, Louisville, KY, 40202 → 100 N Capitol Ave, Lansing, MI, 48933
$1,834.00
315.902 mi
Freight Load 36
100 N Capitol Ave, Lansing, MI, 48933 → 151 W Jefferson, Louisville, KY, 40202
$2,662.00
315.902 mi
Freight Load 43
Dashboard
Stats
Past Loads
Account
Settings
About


12:09
Load Details
Freight Load 21
Load #1
Origin
600 Woodward Ave, Detroit, MI, 48226
Destination
1 Monument Cir, Indianapolis, IN, 46204
Cost
$1,644.00
Distance
240.086 mi
Status
DELIVERED
Dispatcher
Zane Salinas
Created
Dec 2, 2025
Picked Up
Oct 11, 2025
Delivered
Oct 11, 2025
View Route on Maps


12:09
Saginaw
Muskegon
Catalyna Clothing, 18058, Detroit, MI 48...
STI Group, One American Square, India...
Kalamazoo
4 hr 55 min
Toledo
4 hr 28 min
DIANA
Muncie
Springfield
4 hr 47 min
Greenwood
Hamilton
Drive
4 hr 28
7 hr 38
4 days
1 day
4 hr 28 min (289 mi)
Fastest route now, avoids road closures
Preview
Add stops
Share


12:09
Account
Email
Test6@gmail.com
First Name
John
Last Name
Ortiz
Code
+1
Phone Number
(857) 867-1942
Dashboard
Stats
Past Loads
Account
Settings
About
12:10
Settings
Preferences
Distance Unit
Miles
Language
English
Dashboard
Stats
Past Loads
Account
Settings
About