Technical Documentation For

Product Name: Uber

Version: 1.0.0

Date: 🗓 Jun 21, 2025

Product Overview 👀

The application supports real-time ride requests, live tracking, in-app payments, and user account management. The project is developed as part of a university assignment with a strong focus on clean architecture, maintainability, and the implementation of standard design patterns.

User & Account Management:

This module serves as the foundational identity and access management system for the entire application. It is designed to handle the complete user lifecycle for three distinct roles: Passengers, Drivers, and Admins. The core function of this module is to provide a secure, centralized system for user registration, authentication, profile management, and administrative oversight.

Ride Lifecycle Management:

This module looks over and serves as the complete journey of a ride request from initiation to post-ride resolution, thus benefiting both passengers and drivers, while also integrating admin oversight functionalities. It ensures seamless and dynamic coordination between system actors and provides tools for exception handling, ride tracking, and dispute resolution.

Payment & Billing Management:

This module handles the complete payment process, from ride fare calculation to secure payment processing. It ensures smooth transactions for passengers and drivers, supports refund management, and provides admin tools for dispute resolution. The system guarantees accurate billing, tracks ride payment history, and integrates promotional offers, ensuring a transparent and efficient financial experience for all users.

Admin Dashboard & System Analytics:

This module provides administrators with a centralized control hub to oversee platform operations, monitor performance, and maintain system integrity. It equips admins with tools to view real-time system activity, generate analytical reports, manage user issues, and configure platform-wide settings. By offering full visibility into ride data, user behavior, and system health, this module supports effective decision-making and ensures operational efficiency at scale.

Product Objectives 🎯

User & Account Management:

- **Establish Secure Access:** To provide a secure and reliable authentication system that protects user accounts through robust login and multi-factor authentication mechanisms.

- **Enable Seamless Onboarding:** To offer a straightforward registration and verification process for new Passengers and Drivers.
- **Empower User Self-Service:** To give users the ability to manage their own account details, including editing profiles and resetting passwords.
- **Provide Robust Administrative Control:** To equip administrators with tools to manage the user community through features like account suspension and login auditing.

Ride Lifecycle Management:

- **Ensure End-to-End Ride Flow: P**rovide a seamless experience from ride request to completion by enabling clear transitions through all stages of a ride—scheduling, acceptance, navigation, and finalization.
- **Enhance Real-Time Interactions: F**acilitate instant communication and dynamic updates between passengers and drivers, such as location adjustments and live status changes.
- **Support Advanced Booking Scenarios: O**ffer ride scheduling capabilities that allow users to plan future rides, improving user convenience and ride availability forecasting.
- **Enable Transparent Fare Processing:** Accurately calculate fares using real-time metrics like distance, time, and surge pricing, while ensuring payment is handled securely and promptly.
- **Empower Feedback and Rating Systems:** Collect and manage mutual feedback between drivers and passengers to foster trust, improve service quality, and surface performance trends.

Payment & Billing Management:

- Ensure Seamless Payment Flow: Provide a smooth process from ride request to payment completion by enabling clear stages of payment, from fare calculation to processing through a secure payment gateway.
- Enable Convenient Ride Cancellations: Allow passengers to easily cancel or modify their rides, while ensuring fare adjustments and refunds are handled

promptly and accurately.

- Facilitate Real-Time Payment Updates: Ensure passengers and drivers are immediately notified of payment status changes, including payment confirmation, ride fare calculation, and refunds.
- Offer Advanced Payment Management: Provide users with the ability to update payment methods, manage payment history, and handle refunds for discrepancies, offering greater control over their financial transactions.
- Support Dispute Resolution and Refund Processing: Ensure transparency and fairness in managing disputes, allowing admins to intervene when necessary and issue refunds in case of errors or service issues.
- Promote Financial Transparency: Enable users to track ride history, receipts, and earnings, allowing both drivers and passengers to monitor their financial activities and ensuring clarity in every transaction.
- Maintain Secure Payment Gateways: Ensure that all financial transactions are processed securely, utilizing trusted payment gateways to safeguard user data and prevent unauthorized access.

Admin Dashboard & System Analytics:

- **Gain Operational Visibility:** Access system-wide ride statistics and real-time system maps to monitor platform performance, geographic demand, and operational bottlenecks at a glance.
- **Optimize Decision-Making with Reports:** Generate comprehensive reports on revenue, user activity, and audit logs to drive data-informed decisions and identify trends across riders and drivers.
- **Ensure Service Integrity:** Manage disputes and support tickets with full visibility into case details, enabling fast resolution and accountability for both users and support teams.
- **Maintain System Configuration:** Oversee and adjust system tariffs, application settings, and promotional codes to align with business strategies, seasonal demands, and market dynamics.

- **Safeguard System Health:** Monitor backend services for uptime, performance anomalies, and failures, ensuring system stability and proactive incident management.

Product Features 🧩

💡 User & Account Management:

- Core Features (For Passengers and Drivers):
  - ✓ Account Registration
  - ✓ Secure Login and Logout
  - ✓ Email & Phone Number Verification
  - ✓ Self-Service Password Reset
  - ✓ Profile Editing
  - ✓ Account Deletion
  - ✓ Multi-Factor Authentication (MFA)
- Administrative Features:
  - ✓ User List Viewing
  - ✓ Detailed User Profile Viewing
  - ✓ User Account Suspension
  - ✓ Login History Auditing

💡 Ride Lifecycle Management:

- Core Passenger Features:
  - ✓ **Request Ride**: Passengers can instantly request a ride by specifying pickup and drop-off locations. If the driver has not yet accepted, users can update these details in real time.
  - ✓ **Schedule Ride (Future Booking)**: Users can book rides in advance, specifying the desired time and location. This ensures availability

during high-demand periods.

- ✓ **Cancel Ride**: Allows cancellation before or during the ride. Refund policies apply based on cancellation timing and ride status.
- ✓ **Contact Driver**: Provides temporary in-app or masked contact channels between the passenger and assigned driver for coordination.
- ✓ **Report Issue**: After or during a ride, users can report problems including driver behavior, route changes, or unsafe conditions.
- ✓ **View Ride History**: Enables passengers to browse past rides, including time, fare breakdown, driver details, and route taken.
- ✓ **Rate Ride Participant**: Allows passengers to rate and provide feedback for the driver to help maintain service quality.

- Core Driver Features:
  - ✓ **Accept/Reject Ride Request**: Drivers receive real-time ride requests and can choose to accept or decline based on availability or proximity.
  - ✓ **Navigate to Pickup Point**: Integrates with map services to guide drivers efficiently to the pickup location.
  - ✓ **Start Ride / End Ride**: Controls the state of the ride and triggers fare calculations. “Start” marks the beginning of the journey, and “End” confirms completion.
  - ✓ **Pause Ride** *(Extension)*: Allows temporary halting of a ride in exceptional cases (e.g., passenger request, emergencies).
  - ✓ **Calculate Fare**: Fare is dynamically calculated using predefined parameters such as distance, time, surge pricing, and applicable discounts.
  - ✓ **Rate Ride Participant**: Drivers can also rate passengers, promoting mutual accountability.

💡 Payment & Billing Management:

- Core Passenger Features:
  - ✓ **Process Payment:** Passengers can pay for rides seamlessly through integrated payment methods. Payment processing is handled securely via the payment gateway.
  - ✓ **Cancel Ride:** Passengers can cancel a ride before or during the trip. Refunds are processed based on timing and cancellation policies.
  - ✓ **Rate Ride Participant:** After the ride, passengers can rate the driver, contributing to service quality and mutual feedback.
  - ✓ **View Ride History:** Passengers can view their past rides, including detailed fare breakdowns, receipts, and any payment-related issues.
  - ✓ **Report Issue:** In case of billing discrepancies or payment-related concerns, passengers can report issues for resolution and follow up through the app.

- Core Driver Features:
  - ✓ **Process Ride Fare:** Drivers can see the calculated fare for each ride, based on distance, time, and applicable surge pricing.
  - ✓ **View Rider Earnings:** Drivers can track their earnings from completed rides, including tips and payment details.
  - ✓ **Rate Ride Participant:** Drivers can rate passengers after the ride to ensure a fair and transparent feedback system.
  - ✓ **Add/Remove Payment Method:** Drivers can update or remove their payment methods for earnings payouts, ensuring secure transactions.
  - ✓ **Report Issue:** Drivers can report payment or fare-related issues, including discrepancies in fare calculation or payment delays.

- Core Admin Features:

- ✓ **Add New Driver:** Admins can add new drivers to the system, ensuring they are set up with valid payment methods to receive earnings.
- ✓ **Issue Refunds:** Admins have the authority to issue refunds in case of ride disputes, billing errors, or customer complaints.
- ✓ **Suspend Driver:** Admins can suspend drivers if they are found to be violating payment policies or involved in fraudulent activities.
- ✓ **View Ride Logs:** Admins can access detailed ride logs to track and manage any billing issues, discrepancies, or payment histories.
- ✓ **Process Payment via Payment Gateway:** Admins can ensure the secure processing of all payments via an external payment gateway, facilitating the transaction completion for passengers and drivers alike.
- ✓ **Manage Promo Codes:** Admins can create and manage promotional discounts, ensuring they are correctly applied during payment processing.

💡 Admin Dashboard & System Analytics:

- Core Admin Features:
  - ✓ **View System-Wide Ride Statistics:** Analyze ride volume, duration, and usage trends across the platform.
  - ✓ **Generate Reports:** Export reports on revenue, user activity, and system performance.
  - ✓ **Monitor Real-Time System Map:** Track active rides, drivers, and demand areas live on the map.
  - ✓ **View User Audit Logs:** Track user actions like logins, ride requests, and changes.
  - ✓ **Monitor System Health:** Observe platform uptime, errors, and performance metrics.
  - ✓ **Manage Promo Codes:** Create, update, and monitor discount offers.

- ✓ **Manage Disputes and Support Tickets:** Review ride details and resolve user-reported issues.
- ✓ **Configure System Tariffs and Settings:** Adjust fare rules, surge pricing, and system parameters.

Code Snippets

Data Transfer Object (DTO) for Registration Data:

```
// Represents the data required from a user to register.
public record RegisterUserDto(
    string FirstName,
    string LastName,
    string Email,
    string Password,
    UserRole Role // Enum: e.g., Passenger, Driver
);

public enum UserRole { Passenger, Driver }

```

This is an example response with a specific error code:

```
{
   "code": "conflict",
   "message": "Conflict",
   "metadata": {
       "error": "TRANSACTION_ALREADY_PROCESSED",
       "type": "Deposit",
       "uri": "//payments/deposits/af9d61aa-b290-4849-b0a7-77bd85
   }
}
```

Admin Resolving a Support Ticket

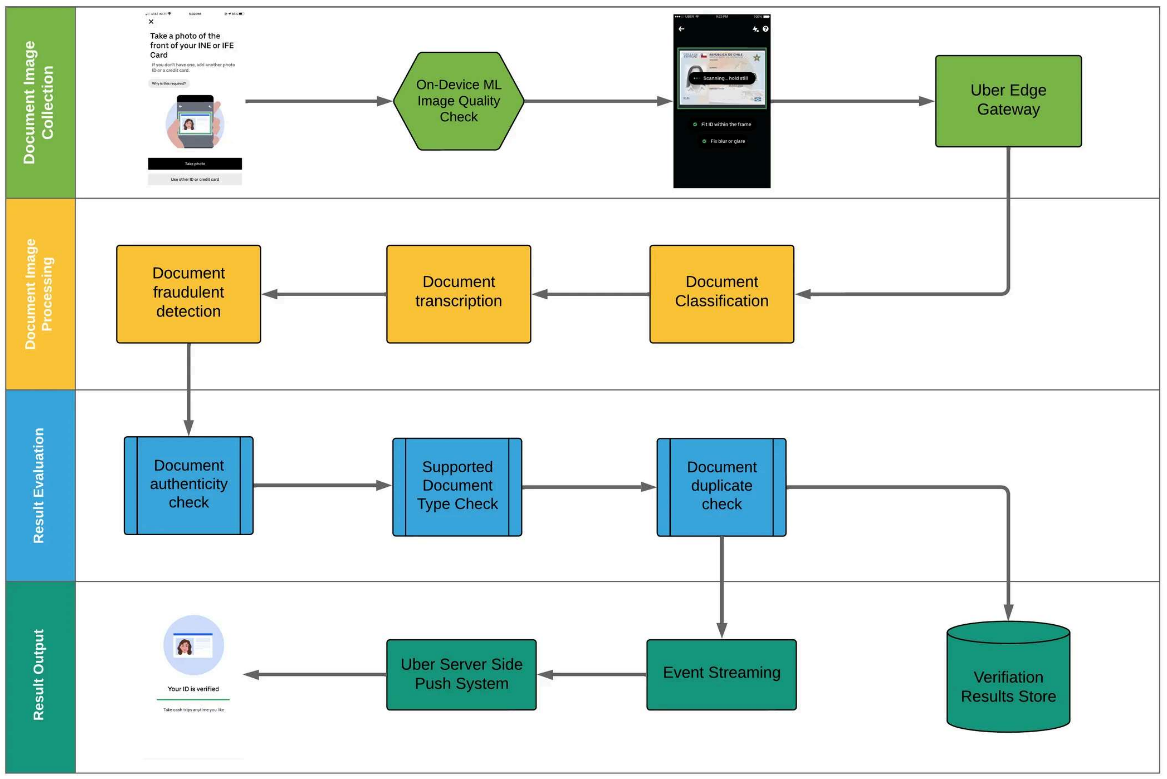
```
// DTO for resolving a support ticket
public record ResolveTicketDto(
    Guid TicketId,
    string ResolutionMessage,
    string ResolvedByAdminId
);
```

Example usage of the ResolveTicketDto in code

```
var resolveRequest = new ResolveTicketDto(
    TicketId: Guid.Parse("c2f9d4a3-76aa-4f7d-aab0-b93b8cd2b1a3"),
    ResolutionMessage: "Issue resolved. Refund processed.",
    ResolvedByAdminId: "admin-1234"
);
```

Some Useful Sources

Diagram on Uber’s Real Time Document Check functionality.


Document Image Collection
Take a photo of the front of your INE or IFE Card
On-Device ML Image Quality Check
Scanning... hold still
Fit ID within the frame
Fix blur or glare
Uber Edge Gateway
Document Image Processing
Document fraudulent detection
Document transcription
Document Classification
Result Evaluation
Document authenticity check
Supported Document Type Check
Document duplicate check
Result Output
Your ID is verified
Uber Server Side Push System
Event Streaming
Verifiation Results Store

API Section 🛠️

🔒 Authentication APIs

POST /api/register

- Registers a new user (driver or passenger)
- Request body: email, password, userType
- Response: JWT token

POST /api/login

- Logs in an existing user
- Request body: email, password
- Response: JWT token

📍 Location APIs

GET /api/nearby-drivers

- Returns a list of available drivers near a user's current location
- **Query params:** `latitude`, `longitude`
- Response: List of drivers with ETA

Third-party API: Google Maps API

- Used for real-time location tracking and route calculation
- **Purpose:** Geocoding, Directions, Distance Matrix

🚗 Ride Management APIs

POST /api/request-ride

- Requests a ride from passenger to destination
- **Body:** `pickup_location`, `dropoff_location`, `passenger_id`
- Response: Assigned driver details and estimated time

POST /api/accept-ride

- Used by drivers to accept ride requests
- **Body:** `ride_id`, `driver_id`

POST /api/complete-ride

- Marks ride as completed and calculates fare

💳 Payment APIs

Third-party API: Stripe API

- Used for secure payment processing
- **Purpose:** Tokenize cards, charge passengers, send payouts to drivers

🔔 Notification APIs

Internal API: POST /api/send-notification

- Sends push or SMS notifications to users
- **Body:** `user_id`, `message`, `type`

Third-party: Firebase Cloud Messaging (FCM)

- Used for real-time push notifications

Code for API's 🔢

```
// File: RegisterUserRequest.cs
public class RegisterUserRequest
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    [MinLength(8)]
    public string Password { get; set; }

    [Required]
    public string FirstName { get; set; }

    [Required]
    public string LastName { get; set; }

    // Can be "Passenger" or "Driver"
    [Required]
    public string Role { get; set; }
}

```

```
// File: LoginRequest.cs
public class LoginRequest
{
    [Required]
    [EmailAddress]
    public string Email { get; set; }

    [Required]
    public string Password { get; set; }
}

// File: AuthResponse.cs
public class AuthResponse
{
    public string UserId { get; set; }
    public string Token { get; set; }
    public DateTime ExpiresAt { get; set; }
}
```

C# API Controller:

This class defines the actual endpoints (/api/auth/register, /api/auth/login) and handles the HTTP requests.

```
// File: AuthController.cs
[ApiController]
[Route("api/[controller]")]
public class AuthController : ControllerBase
{
    // This would be injected via Dependency Injection
    private readonly IAuthService _authService;

    public AuthController(IAuthService authService)
    {
            _authService = authService;
        }

        /// <summary>
        /// Registers a new user (Passenger or Driver).
        /// </summary>
        [HttpPost("register")]
        public async Task<IActionResult> Register([FromBody] Register
        {
            // Logic to create user would go here
            var userId = await _authService.RegisterUserAsync(request
            // After creation, log the user in to get a token
            var response = await _authService.LoginAsync(new LoginRec
            return Ok(response);
        }

        /// <summary>
        /// Logs in an existing user and returns a JWT token.
        /// </summary>
        [HttpPost("login")]
        public async Task<IActionResult> Login([FromBody] LoginReques
        {
            var response = await _authService.LoginAsync(request);
            if (response == null)
            {
                return Unauthorized("Invalid credentials.");
            }
            return Ok(response);
        }
    }
```

Database Schema Design

Sample schema for the core entities: User, Ride, and Vehicle.

```
// File: User.cs
public class User
{
    public Guid Id { get; set; } // Using Guid for unique IDs
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Email { get; set; }
    public string HashedPassword { get; set; }
    public string Role { get; set; } // "Passenger" or "Driver"
    public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

    // Navigation property for rides
    public virtual ICollection<Ride> RidesAsPassenger { get; set;
    public virtual ICollection<Ride> RidesAsDriver { get; set; }
}

// File: Ride.cs
public class Ride
{
    public Guid Id { get; set; }
    public Guid PassengerId { get; set; }
    public Guid? DriverId { get; set; } // Nullable for when ride

    public string Status { get; set; } // e.g., "Requested", "Acc

    // Location data
    public double PickupLatitude { get; set; }
    public double PickupLongitude { get; set; }
    public string PickupAddress { get; set; }
    public double DropoffLatitude { get; set; }
    public double DropoffLongitude { get; set; }
    public string DropoffAddress { get; set; }

    public DateTime RequestedAt { get; set; }
    public DateTime? CompletedAt { get; set; }

    public decimal Fare { get; set; }

    // Navigation properties
    public virtual User Passenger { get; set; }
    public virtual User Driver { get; set; }
}

// File: Vehicle.cs - A driver would have a vehicle
public class Vehicle
{
    public Guid Id { get; set; }
    public Guid DriverId { get; set; } // Link to the User with "
    public string Make { get; set; }
    public string Model { get; set; }
    public string LicensePlate { get; set; }
    public int Year { get; set; }

    // Navigation property
    public virtual User Driver { get; set; }
}
```

Security and Error Handling

we will use JWT Validation for Security

```
// In your main Program.cs or Startup.cs
// This code adds JWT authentication services to the application

```

```
var jwtSettings = builder.Configuration.GetSection("JwtSettings")
var secretKey = Encoding.ASCII.GetBytes(jwtSettings["Secret"]);

builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = JwtBearerDefaults.Authent
    options.DefaultChallengeScheme = JwtBearerDefaults.Authentica
})
.AddJwtBearer(options =>
{
    options.RequireHttpsMetadata = false; // Set to true in produ
    options.SaveToken = true;
    options.TokenValidationParameters = new TokenValidationParame
    {
        ValidateIssuerSigningKey = true,
        IssuerSigningKey = new SymmetricSecurityKey(secretKey),
        ValidateIssuer = false, // Set to true and configure in p
        ValidateAudience = false, // Set to true and configure in
    };
});
```

protect your endpoints with the `[Authorize]` attribute.

```
// In a RideController.cs
[HttpPost("accept-ride")]
[Authorize(Roles = "Driver")] // Only users with the "Driver" rol
public async Task<IActionResult> AcceptRide([FromBody] AcceptRide
{
    // ... logic to accept ride
    return Ok();
}
```

Technical Support Contact Information 💬

Mariam Rusishvili - ✉ mariam.rusishvili.1@iliauni.edu.ge

Nini Jakhaia - ✉ nini.jakhaia.1@iliauni.edu.ge

Tamar Kvirikashvili - ✉ tamar.kvirikashvili.1@iliauni.edu.ge

Tamari Tateshvili - ✉ tamari.tateshvili.1@iliauni.edu.ge