Starter Report (Prototype)

Moudhaffer BOUALLEGUI

Ali BOUHDIBA

Wassim GHAMGUI

Eya JARY

Everyone : Environment & Core Platform Setup

1. **Tool Installation & Repo Setup**
   - Install and configure Python, Flask, MySQL, Git.
   - Initialize the Git repository with a sensible branching model.
2. **Database Design**
   - Define the schema
   - Set up the MySQL database
3. **Basic Features**
   - Build the initial Flask app structure

Ali+Moudhaffer : Secure Authentication & Role Management

1. **Authentication System**
   - Integrate Flask-Login for user sessions
   - Build registration/login flows, password hashing
2. **Authorization & Roles**
   - Enforce Rôle Permissions on all routes
   - Create middleware to guard sensitive endpoints.
3. **Exception Handling**
   - Global error handlers for authentication/authorization failures
   - Securely log stack traces or sensitive info

Eya+Moudhaffer : Input Validation & Data Protection

1. **Input Validation**
   - Implement server-side validation for all forms/APIs (WTForms).
   - Sanitize inputs to prevent SQL injection and XSS.
2. **Data Encryption**
   - Encrypt sensitive fields in the database (personally identifiable info) at rest.
   - Manage encryption keys securely (environment variables).
3. **Secure Exception Handling**
   - Wrap critical database/encryption operations in try/except blocks.
   - Ensure no sensitive data leaks in error messages.

Wassim : Quality Assurance, Testing & Documentation

1. **Automated Code Reviews**
    - Configure SonarLint and Flake8 in the CI pipeline (pre-commit hooks or GitHub Actions).
    - Establish and enforce style/security rules.
2. **Security Testing**
    - Write and run pentests (ZAP).
    - Patch any discovered vulnerabilities.

1 Project Architecture

Layer	Technology	Characteristics
Presentation / UI	• HTML + Jinja2 templates • Bootstrap-inspired CSS kits	– Page flow (home → signup / login → booking → payment → invoice) – How templates are reused (e.g., flash-banner partial, base.html)
Application	**Flask** (single `main.py`)	– Blueprint-like grouping by feature (auth, booking, admin, etc.) – Validators in separate `*_validation.py` modules to keep routes clean
Persistence	MySQL via MySQLdb	– Connection pooling kept simple (global `conn`) for the prototype
Services / Utilities	• Flask-Mail (MailHog dev SMTP) • WeasyPrint (PDF invoices) • Custom `crypto_utils.py`	– Diagram how helpers (mail, pdf, crypto) plug into routes

car_rental/

├── .env

├── .gitattributes

├── app/

│ ├── main.py

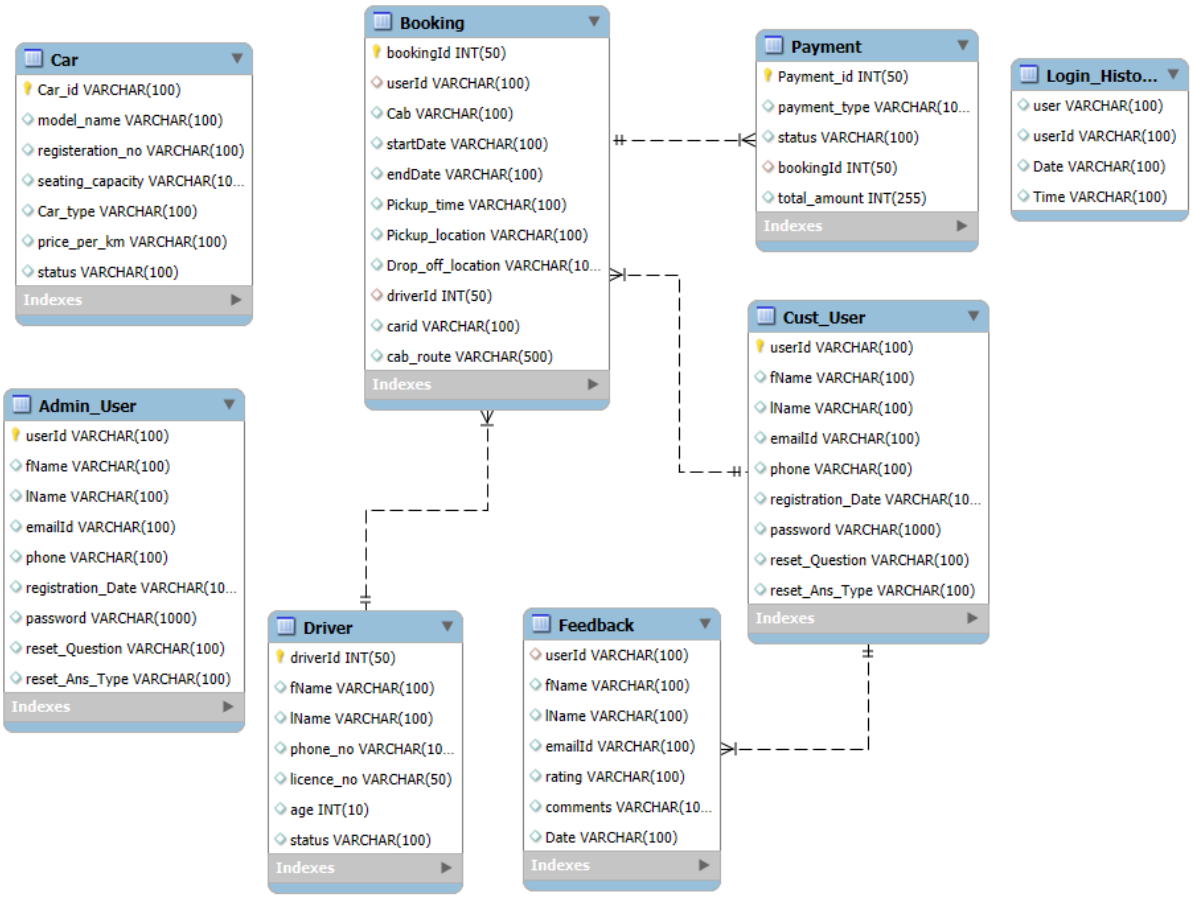
│ ├── requirements.txt

```
├── static/
└── templates/
    ├── about.html
    ├── addadmin.html
    ├── addcar.html
    ├── addcustomer.html
    ├── adddriver.html
    ├── admindetails.html
    ├── adminpage.html
    ├── allbooked.html
    ├── booking.html
    ├── changecarstatus.html
    ├── changedriverstatus.html
    ├── contact.html
    ├── deleteadmin.html
    ├── deletecars.html
    ├── deletedriver.html
    ├── deleteuser.html
    ├── displaybooking.html
    ├── displaycars.html
    ├── displaycustomers.html
    ├── displaydrivers.html
    ├── displaystatuscar.html
    ├── displaystatusdriver.html
    ├── feedback.html
    ├── feedbackdisplay.html
    ├── final.html
    ├── index.html
```

```
│   ├── invoice.html
│   ├── loginhistory.html
│   ├── payment.html
│   ├── resetpassword.html
│   ├── signin.html
│   ├── signup.html
│   └── Status.html
├── db/
│   └── car_rental_db.sql
├── docker-compose.yml
├── Dockerfile
└── README.md
```

2 Database Schema & Data Structure

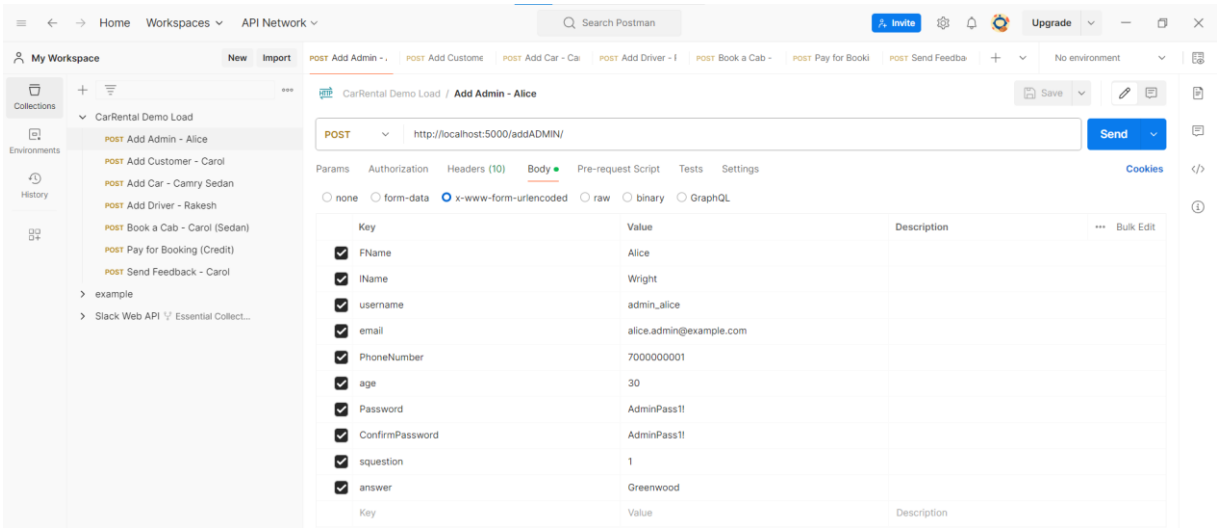
Table	**Key columns**	**Notes**
Cust_User	userId PK	hashed password, encrypted security answer
Admin_User	idem	same structure as Cust_User
Car	Car_id PK	status ENUM(Available/Booked)
Driver	driverId PK	status likewise
Booking	bookingId PK	links user, car, driver, route
Payment	payment_id PK	metadata only – no PAN stored
Feedback	ratings + comments	
Login_History	audit trail	

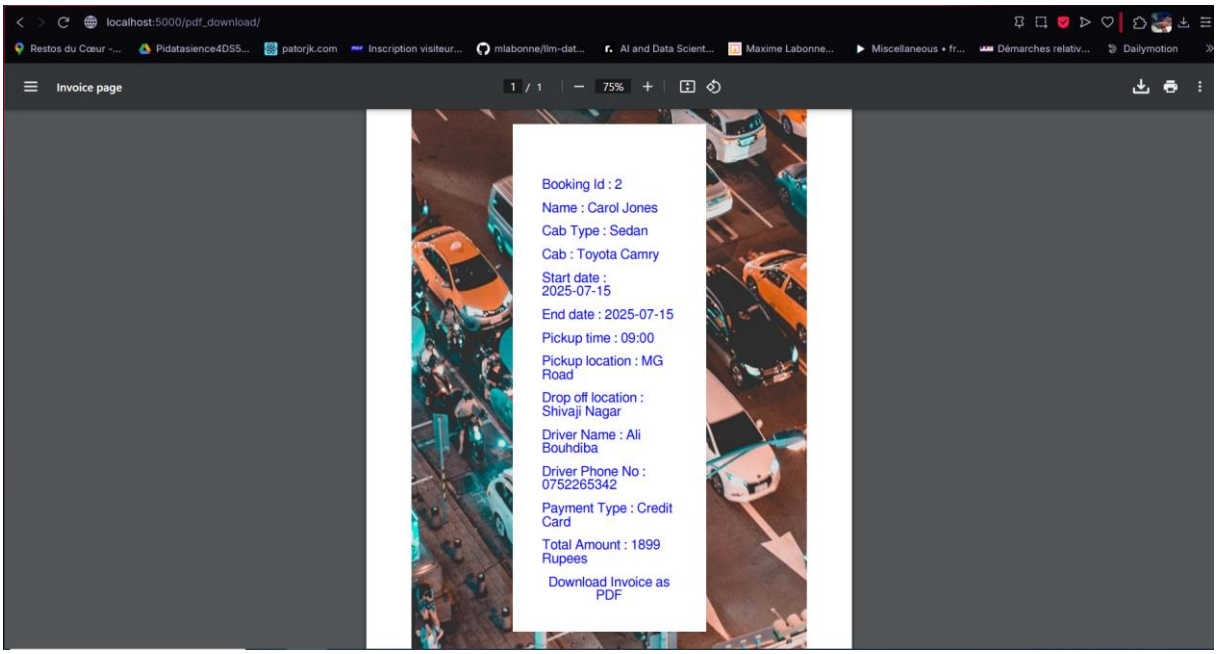


3 Dummy Data

We seeded :

- **2 Admins** (`admin_alice`, `admin_bob`)
- **3 Customers** (`carol_j`, `davidl`, `emma_s`)
- **4 Cars** (Sedan, Hatchback, SUV)
- **3 Drivers**
- **1 Booking** + corresponding **Payment** + **Invoice**



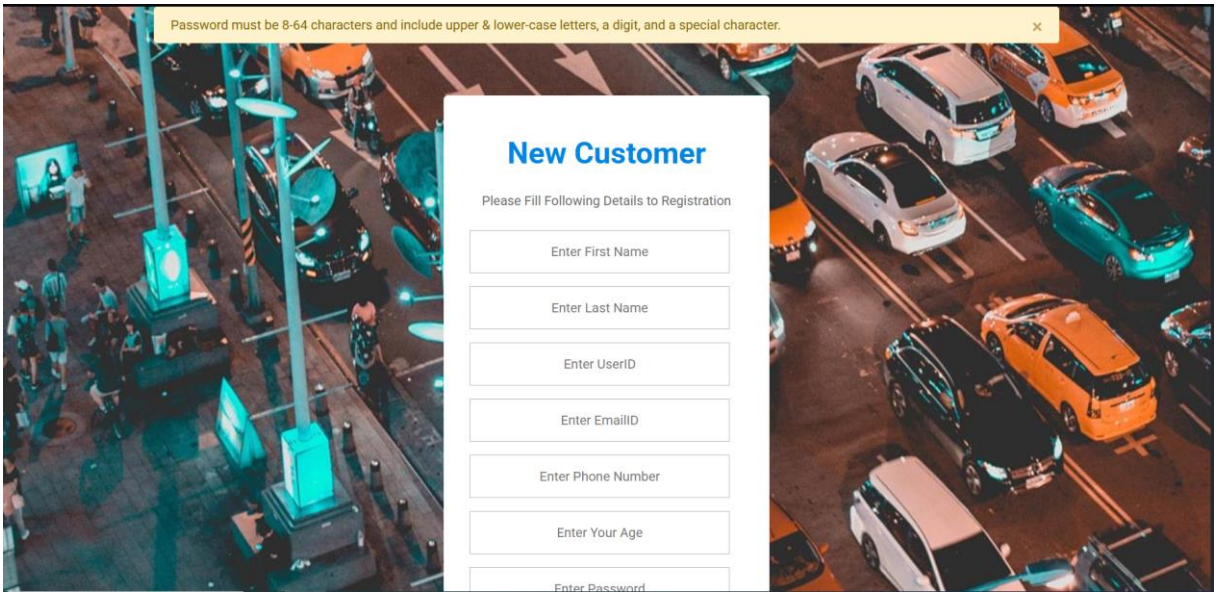


4 Implemented Functionalities

Role	Feature
Visitor	Landing & About pages
Customer	Registration → Login → Booking → Payment → Invoice (PDF) → Feedback
Admin	Add/List/Delete users, cars, drivers; change statuses; dashboards
System	Sends e-mails, records login history, calculates revenue & most-used route

5 Security Measures in the Prototype

Area	Implementation
Passwords	PBKDF2-SHA256 via `passlib` (configurable rounds)
Security Q&A	**AES-128-CBC** with random IV (`encrypt_answer`, `decrypt_answer`) – key loaded from the `AES_SECRET_KEY` environment variable.
Input Validation	Dedicated validator classes for **every** form (`RegistrationForm`, `BookingForm`, `LoginForm`, etc.) – rejects bad input before DB access.
SQL Safety	All queries are parameterized (`%s` placeholders).
Email & Phone *(planned)*	Road-mapped for hashed lookup + encrypted at rest (GDPR alignment).
Session Hardening	Flask's `secret_key` set; no sensitive data in cookies.
Error Handling	Routes return proper HTTP 400/401/404 with flashed messages—prevents information leakage.



6 Next Steps / Road-map

1. **Encrypt remaining PII** (email, phone) with hash-lookup strategy.
2. Move `master_password` and other secrets to a vault.
3. Add CSRF tokens (Flask-WTF).
4. Unit / integration tests (pytest + FactoryBoy).
5. Docker-Compose for one-click spin-up (web, db, mailhog).
6. CI pipeline (GitHub Actions) to lint, test, and build images.