Wheels Sabana – Complete User Stories

Epic	Description	# Tickets	Tipo principal
Registration & Authentication	Registro, login, validación de correo, perfil, logout	6	Front + Back
Vehicles & Driver Management	Vehículos, roles, validación de documentos	5	Front + Back
Trip Management	Creación, reservas, bloqueo, rutas, tarifa	9	Front + Back + Structure
Search & Filters	Filtros de viajes	4	Front
Notifications & Communication	Cancelaciones, avisos, recordatorios	6	Back + Structure
Ratings & Safety	Calificaciones, seguridad, uptime, cifrado	5	Front + Back
Payments (Future)	Métodos de pago y control de ingresos	3	Back
Infra & Performance	Diseño responsive, tiempo de carga, escalabilidad, APIs, sockets	5	Structure + Front + Back

Epic: Registration & Authentication

1. **Title:** Registration with University Email
   Description: As a *student*, I want to register with my institutional email so that I can access the Wheels platform.
   Acceptance Criteria:

- Form accepts first name, last name, university ID, phone and email.
- Email must end with @unisabana.edu.co.
- User saved in DB and password stored hashed.
  Checklist:
- Design registration screen in Figma.
- Implement front-end form and client-side validation.
- Implement backend endpoint POST /auth/register.
- Validate email domain on backend.
- Hash password (bcrypt/argon2) and save user in MongoDB.
- Add success flow → redirect to login and send welcome email.
- Add unit tests for backend validation.
  Labels: Epic: Registration · Frontend · Backend
  Priority: High
  Note: Store minimal PII and follow privacy rules.

---

2. **Title:** Login with Credentials
   Description: As a *registered user*, I want to log in with my email and password so that I can access my account.
   Acceptance Criteria:

- Login accepts email + password and returns auth token.
- Invalid credentials show friendly error.
  Checklist:
- Design login screen in Figma.
- Implement login form (frontend).

- Implement backend POST /auth/login (issue JWT or session).
- Add middleware to protect endpoints.
- Add unit/integration tests.
  Labels: Epic: Registration · Frontend · Backend
  Priority: High

---

3. **Title:** Logout
   Description: As a *user*, I want to log out so that I can protect my account on shared devices.
   Acceptance Criteria:

- Logout clears client token/session.
- Server invalidates refresh tokens if applicable.
  Checklist:
- Add logout button in UI.
- Implement client-side token removal.
- Backend: invalidate refresh token endpoint if using refresh tokens.
- UI redirect to login after logout.
  Labels: Epic: Registration · Frontend · Backend
  Priority: Medium

---

4. **Title:** Password Recovery
   Description: As a *user*, I want to recover my password so that I can regain access if I forget it.
   Acceptance Criteria:

- Recovery request sends single-use link to university email.
- Link expires after a short time.
  Checklist:
- Design password recovery screens (request & reset) in Figma.
- Implement POST /auth/forgot password to send tokenized email.
- Implement POST /auth/reset-password to set new password.

- Secure tokens and expiration in DB.
- Email templates and test flows.
  Labels: Epic: Registration · Frontend · Backend
  Priority: Medium

---

5. **Title:** View & Edit Profile
   Description: As a *user*, I want to view and edit my profile so that I can keep my information up to date.
   Acceptance Criteria:

- User can view profile fields and update permitted fields.
- Changes persist in DB.
  Checklist:
- Design profile screen in Figma (view & edit states).
- Implement frontend profile page and edit form.
- Implement backend GET /users/me and PUT /users/me.
- Validate inputs and save to DB.
- Add profile photo upload handling (storage).
  Labels: Epic: Registration · Frontend · Backend · Structure (storage)
  Priority: High

---

6. **Title:** Validate Institutional Email (System Rule)
   Description: As a *system*, only allow registrations with @unisabana.edu.co to ensure authentic users.
   Acceptance Criteria:

- No account created for other domains.
- Error explains requirement to user.
  Checklist:
- Add server-side email domain validation.
- Add client-side hint to registration form.

- Write tests for domain validation.
  Labels: Epic: Registration · Backend
  Priority: High

Epic: Vehicles & Driver Management

7. **Title:** Register Vehicle
   Description: As a *passenger*, I want to register a vehicle (plate, brand, model, capacity, SOAT, license) so that I can become a driver.
   Acceptance Criteria:

- Form accepts vehicle details and links vehicles to user.
- Mandatory documents upload fields present.
  Checklist:
- Design vehicle registration form in Figma.
- Implement frontend vehicle form and file uploads.
- Implement backend POST /vehicles and DB model.
- Store files (SOAT, license) are securely in storage (e.g., S3).
- Validate mandatory fields and return success.
  Labels: Epic: Vehicles · Frontend · Backend · Structure
  Priority: High

---

8. **Title:** Switch between Passenger and Driver Roles
   Description: As a *user*, I want to switch between passenger and driver so that I can use both modes.
   Acceptance Criteria:

- Toggle in profile to switch mode.
- If no vehicle, switch to driver is disabled with explanation.
  Checklist:
- Design role toggle UI in Figma.
- Implement toggle on frontend and role state.
- Backend: update user role and guard driver-only routes.

- Add validation: block switch if no vehicle or invalid docs.
  Labels: Epic: Vehicles · Frontend · Backend
  Priority: Medium

---

9. **Title:** Manage Multiple Vehicles
   Description: As a *driver*, I want to add, edit or delete vehicles so that I can keep my fleet up to date.
   Acceptance Criteria:

- User can add more than one vehicle and set an active vehicle for a trip.
  Checklist:
- UI for list of vehicles, add/edit/delete in Figma.
- Implement CRUD endpoints /vehicles (GET/POST/PUT/DELETE).
- Implement frontend flows to select active vehicle for trips.
- Tests for data integrity on delete (no orphaned trips).
  Labels: Epic: Vehicles · Frontend · Backend
  Priority: Medium

---

10. **Title:** Validate Vehicle Data
    Description: As a *system*, validate vehicle fields (capacity, plate format) before allowing trips.
    Acceptance Criteria:

- Capacity must be > 0 and ≤ reasonable max.
- Plate format validated per local rules.
  Checklist:
- Implement backend validators for vehicle models.
- Add client-side validation messages.
- Unit tests for validators.
  Labels: Epic: Vehicles · Backend
  Priority: Medium

---

11. **Title:** Validate Documents (SOAT & License)
    Description: As a *system*, ensure SOAT and driver's license are valid before allowing trip creation.
    Acceptance Criteria:

- Expired documents prevent drivers from creating trips.
- Driver is notified to renew documents.
  Checklist:
- Store document expiry dates in DB.
- Implement check on trip creation endpoint.
- Add UI warnings in driver dashboard.
- Notify via push/email when documents near expiry.
  Labels: Epic: Vehicles · Backend · Structure
  Priority: High

Epic: Trip Management

12. **Title:** Create Trip (Driver)
    Description: As a *driver*, I want to create a trip with start, destination, route, time, seats and price so passengers can book.
    Acceptance Criteria:

- Trip stored with required fields; seats ≤ selected vehicle capacity.
- Trip visible to passengers after creation.
  Checklist:
- Design trip creation screen in Figma.
- Implement frontend creation form & validation.
- Backend endpoint POST /trips and DB schema.
- Integrate distance/time calculation (Maps API) to suggest tariff.
- Set default status pending or scheduled.
- Add tests & Swagger docs.
  Labels: Epic: Trips · Frontend · Backend · Structure
  Priority: High

13. **Title:** Add Pickup Points (Driver)
Description: As a *driver*, I want to add pickup points, so passengers know where to board.
Acceptance Criteria:

- Pickup points displayed on trip details and map.
- Passengers can select pickup points during reservation.
Checklist:
- Design pickup point UI in Figma.
- Allow adding pickup points when creating/editing trip.
- Store pickup points with coordinates in DB.
- Show pickup points on trip detail map.
- Ensure passenger reservation requires selecting a pickup point.
Labels: Epic: Trips · Frontend · Backend · Structure
Priority: High

---

14. **Title:** Calculate Distance & Estimated Time (System)
Description: As a *system*, I want to calculate distance and ETA using Google Maps so users get accurate trip info.
Acceptance Criteria:

- Distance and ETA stored and displayed in trip details.
- Request rate limits handled gracefully.
Checklist:
- Integrate Google Maps Directions / Distance Matrix APIs.
- Backend service to calculate and cache distances.
- Display distance and ETA in frontend trip details.
- Implement caching (Redis) for repeated queries.
Labels: Epic: Trips · Structure · Backend
Priority: High

---

15. **Title:** Suggest Tariff (System)
    Description: As a *system*, I want to suggest a tariff based on distance, time and inflation so drivers can price fairly.
    Acceptance Criteria:

- System gives suggested price; driver may edit within allowed range.
- Formula documented.
  Checklist:
- Define tariff formula (base + km*rate + min*rate).
- Implement calculation service in backend.
- Provide driver UI to view & adjust suggested tariff within ±20%.
- Store final tariff chosen.
  Labels: Epic: Trips · Backend · Structure
  Priority: High

---

16. **Title:** View Available Trips (Passenger)
    Description: As a *passenger*, I want to see available trips with driver, route, seats, price and time so I can choose.
    Acceptance Criteria:

- Lists only trips with seats > 0 and status active/scheduled.
- Click to view details.
  Checklist:
- Design trip listing cards in Figma.
- Implement frontend list and filters.
- Backend GET /trips with filtering params.
- Real-time updates for seat changes (sockets or firestore).
- Pagination and performance optimizations.
  Labels: Epic: Trips · Frontend · Backend · Structure
  Priority: High

---

17. **Title:** Reserve Seats (Passenger)
Description: As a *passenger*, I want to reserve seats so I secure a place on the trip.
Acceptance Criteria:

- Seats reserved decrease availability immediately.
- Reservation stores passenger, seats count and pickup point.
Checklist:
- Design reservation UI & confirmation in Figma.
- Implement frontend reservation flow.
- Backend POST /reservations and DB model.
- Implement server-side concurrency control (atomic decrement).
- Send confirmation notification to driver & passenger.
Labels: Epic: Trips · Frontend · Backend · Structure
Priority: High

---

18. **Title:** Reserve Multiple Seats (Passenger)
Description: As a *passenger*, I want to reserve multiple seats in one reservation so I can book for friends.
Acceptance Criteria:

- Users can pick number of seats up to available.
- Each seat can have a pickup point assigned.
Checklist:
- UI to select seat quantity and pickup per seat.
- Backend validation for seats quantity.
- Update reservation model to include array of pickup points.
- Tests for multi-seat reservation edge cases.
Labels: Epic: Trips · Frontend · Backend
Priority: Medium

---

19. **Title:** Block Full Trips (System)
Description: As a *system*, mark trips as “Full” when no seats remain to prevent

overbooking.
Acceptance Criteria:

- Trip status updates to full and UI reflects disabled booking.
  Checklist:
- Implement seat counter logic in backend with atomic checks.
- Update trip status when seats == 0.
- Disable reservation action in frontend for full trips.
- Notify driver that trip is full (optional).
  Labels: Epic: Trips · Backend · Structure
  Priority: High

---

20. **Title:** Driver Views Passenger List
    Description: As a *driver*, I want to see the passenger list and pickup points so I can organize pickup order.
    Acceptance Criteria:

- Driver sees confirmed reservations with passenger name, phone and pickup point.
  Checklist:
- Design driver passenger list screen in Figma.
- Backend GET /trips/:id/passengers.
- Implement frontend dashboard for driver.
- Option to export or message passengers (optional).
  Labels: Epic: Trips · Frontend · Backend
  Priority: Medium

Epic: Search & Filters

21. **Title:** Filter by Departure Point
    Description: As a *passenger*, I want to filter trips by departure point (e.g., Puente Madera, Ad Portas) so I only see relevant trips.
    Acceptance Criteria:

- Filter returns only matching trips; UI shows active filter.
  Checklist:
- Add departure point filter UI in Figma.
- Implement backend filter param departure_point.
- Wire frontend filter and update listing query.
- Add unit tests for filter logic.
  Labels: Epic: Search · Frontend · Backend
  Priority: High

---

22. **Title:** Filter by Seats Available
    Description: As a *passenger*, I want to filter trips by minimum seats available so I only see trips I can book for my party.
    Acceptance Criteria:

- Filter accepts minimum seats and updates list.
  Checklist:
- Design seats filter control.
- Implement backend min_seats param.
- Client-side filter options and test.
  Labels: Epic: Search · Frontend · Backend
  Priority: Medium

---

23. **Title:** Filter by Time Range
    Description: As a *passenger*, I want to filter trips by departure time range so I can find trips that fit my schedule.
    Acceptance Criteria:

- Filter by start time and end time and results update accordingly.
  Checklist:
- Time range UI (picker) in Figma.
- Backend start_time & end_time filtering.

- Frontend integration and testing.
  Labels: Epic: Search · Frontend · Backend
  Priority: Medium

---

24. **Title:** Filter by Maximum Price
    Description: As a *passenger*, I want to filter trips by a maximum price, so I only see affordable options.
    Acceptance Criteria:

- Slider or input for max price; results respect the bound.
  Checklist:
- Price filter UI in Figma.
- Backend max_price filter param.
- Frontend binding and test.
  Labels: Epic: Search · Frontend · Backend
  Priority: Medium

Epic: Notifications & Communication

25. **Title:** Trip Cancellation Notification (Driver cancels)
    Description: As a *passenger*, I want to receive immediate notification if my trip is canceled so I can make alternative plans.
    Acceptance Criteria:

- Push notification sent to all booked passengers and email fallback.
  Checklist:
- Design notification message templates.
- Implement backend event trip: cancelled and push/email triggers.
- Implement client-side notification handler (in-app).
- Tests for notification delivery.
  Labels: Epic: Notifications · Backend · Structure
  Priority: High

---

26. **Title:** Trip Time Change Notification
    Description: As a *passenger*, I want to be notified if the driver changes the departure time so I stay updated.
    Acceptance Criteria:

- All booked passengers receive updated time notification.
  Checklist:
- Implement backend event trip: updated with diff detection.
- Send push + email to affected passengers.
- UI shows updated trip time and history.
  Labels: Epic: Notifications · Backend · Frontend
  Priority: Medium

---

27. **Title:** Notify Driver of New Reservation
    Description: As a *driver*, I want to be notified when someone reserves a seat so I can confirm and prepare.
    Acceptance Criteria:

- Driver receives push notification and sees reservation in dashboard.
  Checklist:
- Emit event on successful reservation.
- Push + email to driver.
- Driver trip dashboard update in real time.
  Labels: Epic: Notifications · Backend · Structure
  Priority: High

---

28. **Title:** Passenger Cancels Reservation
    Description: As a *passenger*, I want to cancel my reservation so the seat becomes available for others.
    Acceptance Criteria:

- Seats increment back and driver is notified.
  Checklist:
- UI flow for cancel reservation in Figma.

- Backend DELETE /reservations/:id or state update.
- Release seats atomically and notify driver.
- Refund/acknowledgment message if applicable (cash/Nequi note).
  Labels: Epic: Notifications · Frontend · Backend
  Priority: Medium

---

29. **Title:** Driver Cancels Trip
    Description: As a *driver*, I want to cancel a trip so no more passengers attempt to book and current passengers are informed.
    Acceptance Criteria:

- Trip status becomes cancelled and passengers notified immediately.
  Checklist:
- Driver cancellation UI & confirmation modal.
- Backend PUT /trips/:id/cancel.
- Notify all passengers (push + email).
- Mark reservations with cancelled status.
  Labels: Epic: Notifications · Frontend · Backend
  Priority: High

---

30. **Title:** Trip Reminder Notifications
    Description: As a *system*, I want to send reminders to driver and passengers before trip departure so everyone is punctual.
    Acceptance Criteria:

- Reminders sent 60 and/or 30 minutes prior (configurable).
  Checklist:
- Scheduler service to queue reminders.
- Push & email reminder templates.
- Configurable reminder times in settings.
  Labels: Epic: Notifications · Structure · Backend
  Priority: Medium

Epic: Ratings & Safety

31. **Title:** Rate Driver
    Description: As a *passenger*, I want to rate the driver after a trip so I can provide feedback and help maintain quality.
    Acceptance Criteria:

- Rating (1–5 stars) saved and associated with trip and driver.
  Checklist:
- Star rating UI (modal) in Figma.
- Frontend flow to submit rating post-trip.
- Backend POST /ratings and average calculation.
- Display driver average rating on profile and trips.
  Labels: Epic: Ratings · Frontend · Backend
  Priority: Medium

---

32. **Title:** Rate Passengers (Driver)
    Description: As a *driver*, I want to rate passengers after trips so community trust is maintained.
    Acceptance Criteria:

- Driver can submit ratings for passengers; stored and used for moderation.
  Checklist:
- Driver rating UI.
- Backend endpoint to save passenger ratings.
- Display passenger average rating in profile.
  Labels: Epic: Ratings · Frontend · Backend
  Priority: Low

---

33. **Title:** Display Average Rating on Profiles
    Description: As a *user*, I want to see average ratings on driver/passenger profiles so I can choose trusted partners.
    Acceptance Criteria:

- Average rating visible on profile and trip cards.
  Checklist:
- Add rating field to profile UI.
- Backend aggregation query to compute averages.
- Cache averages for performance (update on new rating).
  Labels: Epic: Ratings · Frontend · Backend · Structure
  Priority: Medium

---

34. **Title:** Encrypt Passwords & Protect PII
    Description: As a *system*, I must encrypt passwords and protect personal data to comply with privacy rules.
    Acceptance Criteria:

- Passwords hashed; PII stored and accessed securely.
  Checklist:
- Use bcrypt/argon2 for passwords.
- Use TLS for all transport.
- Limit PII exposure in APIs; audit logs.
- Data retention policy and delete flows.
  Labels: Epic: Safety · Backend · Structure
  Priority: High

---

35. **Title:** System Availability (Uptime)
    Description: As a *stakeholder*, I want the app to be available ≥99% so students rely on it.
    Acceptance Criteria:

- Monitoring and alerts in place; SLA defined.
  Checklist:
- Setup monitoring (Prometheus/Sentry/NewRelic).
- Setup alerts and on-call runs.

- Define maintenance windows and fallback modes.
  Labels: Epic: Safety · Structure · Backend
  Priority: High

Epic: Payments (Future)

36. **Title:** Cash / Nequi Payment Option (Informational)
    Description: As a *passenger*, I want to pay by cash or Nequi to the driver so I can use the service without in-app payments.
    Acceptance Criteria:

- Payment method recorded on reservation but no real transaction processed.
  Checklist:
- Add payment method selector to reservation flow.
- Show payment instructions to passenger & driver.
- Record payment method in reservation record.
  Labels: Epic: Payments · Frontend · Backend
  Priority: Low

---

37. **Title:** Driver Payment History (Manual)
    Description: As a *driver*, I want to see a history of reservations and manual payments so I can track earnings.
    Acceptance Criteria:

- Driver sees reservations and a field where they mark payment received (cash/Nequi).
  Checklist:
- Design earnings/history screen.
- Backend query for driver reservations and payment status.
- UI toggle to mark payment received.
  Labels: Epic: Payments · Frontend · Backend
  Priority: Low

---

38. **Title:** Online Payments Integration (Future)
    Description: As a *system*, I want the option to integrate in the future with Nequi or similar so we enable in-app payments.
    Acceptance Criteria:

- Architecture allows adding payments provider without major refactor.
  Checklist:
- Design payments abstraction in backend.
- Research Nequi / MercadoPago APIs and compliance.
- Leave hooks in frontend for payment flow.
  Labels: Epic: Payments · Structure · Backend
  Priority: Low

Epic: Infra & Performance

39. **Title:** Responsive Design (UI)
    Description: As a *user*, I want the app to work on phone, tablet and desktop so I can use it anywhere.
    Acceptance Criteria:

- Key screens render correctly on common breakpoints.
  Checklist:
- Define responsive breakpoints and grids in design system.
- Implement responsive CSS/containers.
- Test on mobile and desktop.
  Labels: Epic: Infra · Frontend · Structure
  Priority: High

---

40. **Title:** Page & API Load Times < 2s
    Description: As a *user*, I expect critical screens to load in under 2 seconds for good UX.
    Acceptance Criteria:

- Home/list pages meet LCP/TTI targets; API median latency within target.
  Checklist:

- SSR/SSG strategy (Next.js) for landing pages.
- Add caching (CDN) and API caching (Redis).
- Optimize images & lazy-load maps.
- Add performance monitoring.
  Labels: Epic: Infra · Frontend · Backend · Structure
  Priority: High

---

41. **Title:** Scalable Architecture
    Description: As a *system*, I want architecture that can scale horizontally so the app supports many users.
    Acceptance Criteria:

- Stateless API, scalable DB, and autoscaling configured.
  Checklist:
- Deploy to cloud with autoscaling (Cloud Run / ECS / GKE).
- Use managed MongoDB Atlas and Redis.
- Design health checks and load testing plan.
  Labels: Epic: Infra · Structure · Backend
  Priority: High

---

42. **Title:** API Integrations (Maps, Waze, TransMilenio, optional Uber reference)
    Description: As a *system*, I want to integrate Maps, Waze and TransMilenio data to improve routing and options.
    Acceptance Criteria:

- Maps used for geocoding/directions; Waze deep-link available; TransMilenio data consumed if available.
  Checklist:
- Integrate Google Maps (Key management).
- Implement Waze deep-link and driver option to open Waze.
- Ingest TransMilenio open data for paradas if applicable.

- Create fallback flows if APIs fail (graceful degrade).
  Labels: Epic: Infra · Structure · Backend
  Priority: High

---

43. **Title:** Real-time Updates & Sockets (Seat availability)
    Description: As a *user*, I want seat counts and trip changes to update in real time so I see current availability.
    Acceptance Criteria:

- Seat availability updates instantly on list & detail views.
  Checklist:
- Decide socket strategy (Socket.IO) or Firestore realtime.
- Implement backend socket server or Firestore listeners.
- Emit events on reservation create/cancel and trip update.
- Frontend listeners update UI and show visual change indicators.
  Labels: Epic: Infra · Structure · Backend · Frontend
  Priority: High

Primeras Correcciones

1. Agregar diseños de error en formularios

- Afecta a:

- Registration with University Email
- Login with Credentials
- Password Recovery
- Register Vehicle
- Create Trip (Driver)
- Reserve Seats (Passenger)

CORRECCIÒN:

Checklist:

- Design error states in Figma (invalid inputs, empty fields, wrong credentials).
- Show clear error messages below fields or banners.
- Validate design for both Desktop and Mobile.

AGREGAR TICKET NECESARIO

Epic: Infra & Performance // Notifications & Communication

Title: Error States for Forms (Desktop & Mobile)

Description: As a user, I want clear visual feedback when I make mistakes filling out forms, so that I can understand what went wrong and fix it easily.

Acceptance Criteria:

- All main forms (Registration, Login, Vehicle Registration, Trip Creation, Trip Reservation) display proper error states.
- Fields with errors are visually highlighted (e.g., red border or icon).
- Error messages appear clearly below the affected field or as a banner.
- Consistent typography, colors, and iconography with Wheels' visual style.
- Designs are provided for both Desktop and Mobile.

Checklist:

- Design error states in Figma for each form.
- Add error messages for invalid inputs (email, password, required fields, etc.).
- Ensure consistent layout and visual hierarchy for all messages.
- Export screens and include naming convention: error_state_[form_name].png.
- Review accessibility: contrast and message clarity.
- Validate design with the dev team before integration.

Labels: Epic: Infra & Performance // Notifications & Communication · Structure · Frontend

Priority: High

2. Agregar el contrato de integración (endpoint, verbo, payload)

Afecta a:
- Todas las historias que tengan interacción con backend.
Ejemplo: Auth, Vehicles, Trips, Reservations, Notifications, Ratings, etc.

Corrección: Al final de cada historia, añadir una nueva sección llamada:

API Contract:

- **Endpoint:** `/nombre-del-endpoint`
- **Method:** `GET / POST / PUT / DELETE`
- **SideNote:** Request Payload & Response Payload

Tipo de corrección	Qué hacer	En qué historias
Diseños de error	Agregar pantallas con errores de validación (Desktop + Mobile)	Formularios de login, registro, password recovery, registro de vehículo, creación de viaje, reserva
Contrato de integración	Añadir sección con endpoint, verbo y payload	Todas las historias que tienen backend

Historias que requieren “API Contract”

Historia	Endpoint sugerido	Método	Notas
Registration with University Email	`/auth/register`	**POST**	Registrar nuevo usuario con correo institucional.
Login with Credentials	`/auth/login`	**POST**	Devuelve token JWT y datos del usuario.
Logout	`/auth/logout`	**POST**	Invalida el token o sesión.
Password Recovery	`/auth/forgot-password` y `/auth/reset-password`	**POST**	Primer endpoint envía correo; segundo resetea contraseña.
View & Edit Profile	`/users/me`	**GET / PUT**	Ver y editar información del perfil.
Validate Institutional Email (System Rule)	`/auth/register`	**POST**	Validación dentro del registro (correo con @unisabana.edu.co).
Register Vehicle	`/vehicles`	**POST**	Registro de vehículo y carga de documentos.
Switch between Passenger and Driver Roles	`/users/role`	**PUT**	Cambia rol activo del usuario.
Manage Multiple Vehicles	`/vehicles`	**GET / PUT / DELETE**	CRUD completo de vehículos del usuario.
Validate Vehicle Data	`/vehicles/validate`	**POST**	Valida formato y capacidad.
Validate Documents (SOAT & License)	`/vehicles/documents/validate`	**GET / POST**	Verifica vigencia de documentos.
Create Trip (Driver)	`/trips`	**POST**	Crea nuevo viaje asociado al conductor.
Add Pickup Points (Driver)	`/trips/:id/pickups`	**POST / PUT**	Agregar o editar puntos de recogida.
Calculate Distance & Estimated Time (System)	`/maps/calculate`	**POST**	Servicio interno que usa Google Maps API.

Suggest Tariff (System)	`/trips/tariff/suggest`	**POST**	Calcula tarifa sugerida según distancia/tiempo.
View Available Trips (Passenger)	`/trips`	**GET**	Lista viajes disponibles con filtros.
Reserve Seats (Passenger)	`/reservations`	**POST**	Crear reserva de cupo.
Reserve Multiple Seats (Passenger)	`/reservations`	**POST**	Misma ruta, payload con `seatsCount` o arreglo de pasajeros.
Block Full Trips (System)	`/trips/:id/status`	**PUT**	Cambia estado del viaje a “full”.
Driver Views Passenger List	`/trips/:id/passengers`	**GET**	Lista pasajeros y puntos de recogida.
Filter by Departure Point	`/trips?departure_point=`	**GET**	Filtro de viajes por punto de salida.
Filter by Seats Available	`/trips?min_seats=`	**GET**	Filtro de viajes según cupos disponibles.
Filter by Time Range	`/trips?start_time=&end_time=`	**GET**	Filtrado por hora.
Filter by Maximum Price	`/trips?max_price=`	**GET**	Filtrado por precio máximo.
Trip Cancellation Notification (Driver cancels)	`/trips/:id/cancel`	**PUT**	Cambia estado del viaje a “cancelled”.
Trip Time Change Notification	`/trips/:id`	**PUT**	Actualiza hora y dispara evento de notificación.
Notify Driver of New Reservation	`/notifications/driver`	**POST (system event)**	Se envía cuando hay nueva reserva.
Passenger Cancels Reservation	`/reservations/:id`	**DELETE**	Cancela y libera asiento.
Driver Cancels Trip	`/trips/:id/cancel`	**PUT**	Marca viaje como cancelado y notifica pasajeros.

Trip Reminder Notifications	`/notifications/reminder`	**POST (system event)**	Evento programado antes del viaje.
Rate Driver	`/ratings/driver`	**POST**	Guarda calificación de pasajero hacia conductor.
Rate Passengers (Driver)	`/ratings/passenger`	**POST**	Guarda calificación de conductor hacia pasajeros.
Display Average Rating on Profiles	`/ratings/average/:userId`	**GET**	Devuelve promedio de calificación.
Encrypt Passwords & Protect PII	`/auth/register`	**POST**	Validación interna del backend.
System Availability (Uptime)	`/health`	**GET**	Endpoint interno para monitoreo.
Cash / Nequi Payment Option (Informational)	`/reservations/:id/payment`	**PUT**	Guarda método de pago elegido.
Driver Payment History (Manual)	`/drivers/:id/payments`	**GET**	Consulta pagos registrados manualmente.
Online Payments Integration (Future)	`/payments/checkout`	**POST**	Integración futura con proveedor (Nequi/MercadoPago).
API Integrations (Maps, Waze, TransMilenio)	`/integrations/maps`, `/integrations/transmilenio`	**GET / POST**	APIs externas de transporte.
Real-time Updates & Sockets (Seat availability)	`/socket.io`	**WS / Event**	Canal de eventos en tiempo real.

- **Registration with University Email**
  Endpoint: `/auth/register`
  Method: POST
  Side Note: Registers a new user with institutional email validation (`@unisabana.edu.co`). Passwords must be hashed and non-institutional domains rejected.
- **Login with Credentials**
  Endpoint: `/auth/login`
  Method: POST
  Side Note: Authenticates the user and returns a JWT token. Provide clear error messages for invalid credentials.
- **Logout**
  Endpoint: `/auth/logout`
  Method: POST
  Side Note: Invalidates the current token or session. If refresh tokens are used, revoke them on the server.
- **Password Recovery**
  Endpoint: `/auth/forgot-password` and `/auth/reset-password`
  Method: POST
  Side Note: `/forgot-password` sends a reset link/token to the institutional email; `/reset-password` verifies and updates the password. Tokens must be single-use and expire shortly.
- **View & Edit Profile**
  Endpoint: `/users/me`
  Method: GET / PUT
  Side Note: Allows users to view and update their data (excluding institutional email). Validate all inputs and persist changes securely.
- **Validate Institutional Email (System Rule)**
  Endpoint: `/auth/register`
  Method: POST
  Side Note: Backend rule that ensures only `@unisabana.edu.co` emails are accepted during registration.
- **Register Vehicle**
  Endpoint: `/vehicles`
  Method: POST
  Side Note: Registers a driver's vehicle with required fields (plate, brand, capacity, SOAT, license). Validate formats and store files securely.

- **Switch between Passenger and Driver Roles**
  Endpoint: `/users/role`
  Method: PUT
  Side Note: Updates the user's active role. Prevent switching to driver if no valid vehicle or expired documents.
- **Manage Multiple Vehicles**
  Endpoint: `/vehicles`
  Method: GET / PUT / DELETE
  Side Note: Full CRUD for managing multiple vehicles per user. Prevent deletion if a vehicle is linked to active trips.
- **Validate Vehicle Data**
  Endpoint: `/vehicles/validate`
  Method: POST
  Side Note: Validates capacity, plate format, and logical limits before allowing the vehicle to be used in trips.
- **Validate Documents (SOAT & License)**
  Endpoint: `/vehicles/documents/validate`
  Method: GET / POST
  Side Note: Verifies expiration dates of SOAT and driver's license; block trip creation if expired.
- **Create Trip (Driver)**
  Endpoint: `/trips`
  Method: POST
  Side Note: Creates a new trip linked to a driver and vehicle. Validate that available seats ≤ vehicle capacity and that documents are valid.
- **Add Pickup Points (Driver)**
  Endpoint: `/trips/:id/pickups`
  Method: POST / PUT
  Side Note: Adds or edits pickup points with coordinates. Passengers must select a pickup when booking.
- **Calculate Distance & Estimated Time (System)**
  Endpoint: `/maps/calculate`
  Method: POST
  Side Note: Internal service using Google Maps APIs. Cache repeated distance calculations for efficiency.
- **Suggest Tariff (System)**
  Endpoint: `/trips/tariff/suggest`
  Method: POST

Side Note: Suggests a price based on distance/time. Driver can edit the price within an allowed range.

- **View Available Trips (Passenger)**
  Endpoint: `/trips`
  Method: GET
  Side Note: Lists all available trips with filters (price, departure point, seats, time). Supports pagination and live updates.
- **Reserve Seats (Passenger)**
  Endpoint: `/reservations`
  Method: POST
  Side Note: Books seats and decreases availability atomically to prevent overbooking.
- **Reserve Multiple Seats (Passenger)**
  Endpoint: `/reservations`
  Method: POST
  Side Note: Allows reserving multiple seats in one request; optionally assign pickup points per seat.
- **Block Full Trips (System)**
  Endpoint: `/trips/:id/status`
  Method: PUT
  Side Note: Updates the trip status to "full" when no seats remain. Frontend should disable further bookings.
- **Driver Views Passenger List**
  Endpoint: `/trips/:id/passengers`
  Method: GET
  Side Note: Lists all confirmed passengers with name, phone, and pickup point. Optionally allow exporting or contacting passengers.
- **Filter by Departure Point**
  Endpoint: `/trips?departure_point=`
  Method: GET
  Side Note: Filters trips by departure point (e.g., Puente Madera). The active filter should be visible in the UI.
- **Filter by Seats Available**
  Endpoint: `/trips?min_seats=`
  Method: GET
  Side Note: Filters trips that have at least the specified number of available seats.
- **Filter by Time Range**
  Endpoint: `/trips?start_time=&end_time=`

Method: GET
Side Note: Filters trips by time range; handle time zone and ISO date formatting.

- **Filter by Maximum Price**
  Endpoint: `/trips?max_price=`
  Method: GET
  Side Note: Filters trips by maximum price; can be combined with pagination and sorting.
- **Trip Cancellation Notification (Driver Cancels)**
  Endpoint: `/trips/:id/cancel`
  Method: PUT
  Side Note: Marks the trip as cancelled and sends push/email notifications to all passengers.
- **Trip Time Change Notification**
  Endpoint: `/trips/:id`
  Method: PUT
  Side Note: Updates the departure time and triggers notifications to all affected passengers.
- **Notify Driver of New Reservation**
  Endpoint: `/notifications/driver`
  Method: POST (system event)
  Side Note: Event emitted when a passenger books a seat; sends push/email notification to the driver.
- **Passenger Cancels Reservation**
  Endpoint: `/reservations/:id`
  Method: DELETE
  Side Note: Cancels the reservation, releases the seat, and notifies the driver. Optionally record cancellation reason.
- **Driver Cancels Trip**
  Endpoint: `/trips/:id/cancel`
  Method: PUT
  Side Note: Cancels the trip, notifies passengers, and marks reservations as cancelled.
- **Trip Reminder Notifications**
  Endpoint: `/notifications/reminder`
  Method: POST (system event)
  Side Note: Sends reminders 60 or 30 minutes before departure using background jobs or schedulers.

- **Rate Driver**
  Endpoint: `/ratings/driver`
  Method: POST
  Side Note: Allows passengers to rate the driver after completing the trip.
- **Rate Passengers (Driver)**
  Endpoint: `/ratings/passenger`
  Method: POST
  Side Note: Allows the driver to rate passengers. Only one rating per passenger per trip.
- **Display Average Rating on Profiles**
  Endpoint: `/ratings/average/:userId`
  Method: GET
  Side Note: Returns a user's average rating. Cache results and update when a new rating is added.
- **Encrypt Passwords & Protect PII**
  Endpoint: `/auth/register`
  Method: POST
  Side Note: Apply security best practices — use bcrypt/Argon2, HTTPS/TLS, and limit personally identifiable data in responses.
- **System Availability (Uptime)**
  Endpoint: `/health`
  Method: GET
  Side Note: Simple health check endpoint for monitoring and uptime verification.
- **Cash / Nequi Payment Option (Informational)**
  Endpoint: `/reservations/:id/payment`
  Method: PUT
  Side Note: Saves the selected payment method (cash or Nequi) without processing transactions. Displays payment instructions.
- **Driver Payment History (Manual)**
  Endpoint: `/drivers/:id/payments`
  Method: GET
  Side Note: Returns manually recorded payments per trip, with filters for date and payment status.
- **Online Payments Integration (Future)**
  Endpoint: `/payments/checkout`
  Method: POST
  Side Note: Placeholder for future integration with Nequi or MercadoPago. Design backend abstraction to avoid tight coupling.

- **API Integrations (Maps, Waze, TransMilenio)**
  Endpoint: `/integrations/maps`, `/integrations/transmilenio`
  Method: GET / POST
  Side Note: Interfaces for external APIs (Maps/Waze/TransMilenio). Manage API keys and fallbacks safely.
- **Real-time Updates & Sockets (Seat Availability)**
  Endpoint: `/socket.io` (or similar WS endpoint)
  Method: WebSocket / Event
  Side Note: Real-time channel for updating seat availability and trip changes instantly across connected clients.