Project Report

“ViaVan” application for tracking and booking van

Members

Natpakan Sangwichain - 6422770196

Naphat Siriwong - 6422770204

Thanabordin Akkharachinoret - 6422770329

Matas Thanamee - 6422771251

Woramate Simrum - 6422771400

Gorawit Khovintasets - 6422771657

Perapat Taytad - 6422780229

Apiwit Nathong - 6422782167

Pattarit Sotyom - 6422782589

Pawarit Kongwan - 6422782605

Nat Srisuksai - 6422782837

Sirindhorn International Institute of Technology

Thammasat University

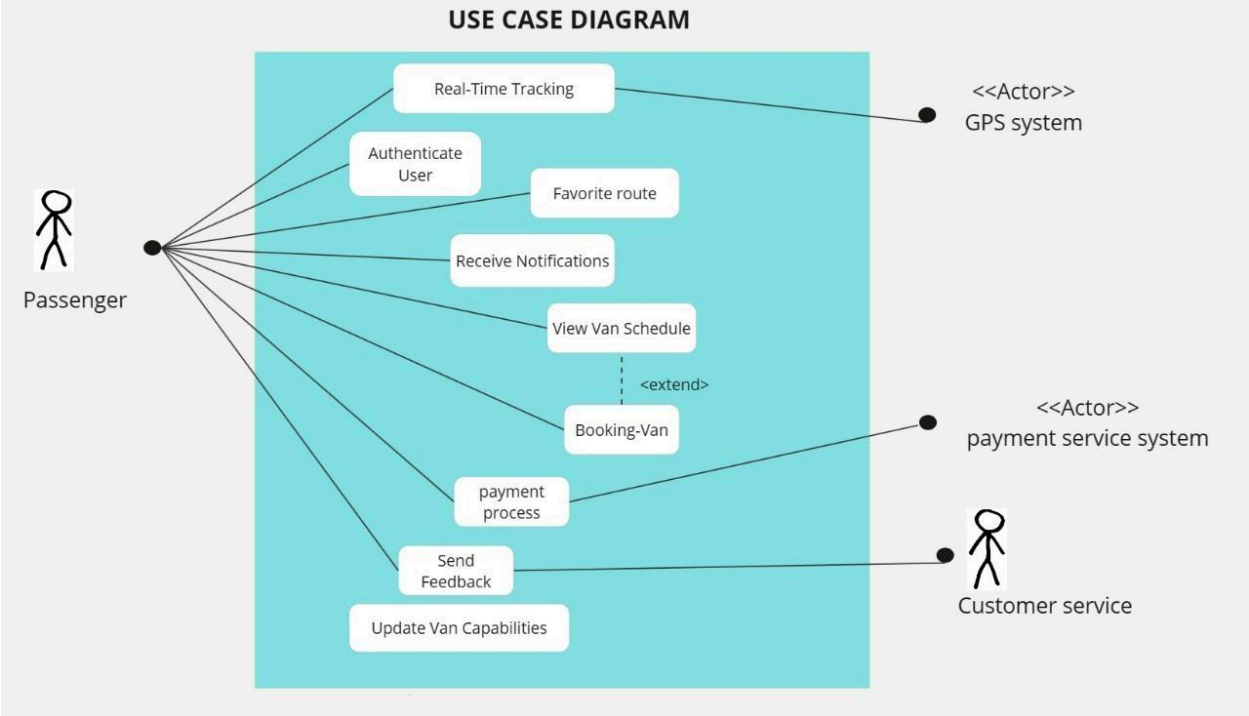
6 May 2024

Introduction

AppViaVan is your ultimate travel companion, transforming the way you experience public transportation. With its cutting-edge technology, this app puts control back in your hands, making van journeys more efficient, reliable, and convenient. Whether you're planning your daily commute or an occasional trip across town, AppViaVan offers a seamless interface to view schedules, track real-time locations, and discover the best routes tailored to your needs.

Beyond just planning and tracking, the app empowers you with the ability to receive instant notifications on service changes, bookmark your favorite routes for quick access, and even book vans directly through your phone. Its interactive platform also allows you to engage directly with the service by reporting issues or providing valuable feedback, ensuring a continuously improving travel experience. AppViaVan is not just an app; it's a smart travel solution designed to enhance public transport for commuters everywhere.

Use case Diagram



Use case Description (Brief format)

Use Case UC1: Authenticate Passenger

- A Passenger attempts to access ViaVan. The system prompts the Passenger to provide credentials (e.g., username and password). The Passenger enters the required information. The system validates the credentials against stored data. If the credentials are correct, the system grants access to the Passenger. If not, access is denied, and the Passenger may be prompted to retry or recover their credentials.

Use Case UC3: Send Notifications

- Passengers schedule appointments and opt to receive notifications through push notifications. As the appointment approaches, the system sends reminders containing essential details such as date, time, and location. Passengers confirm or reschedule appointments upon receiving the reminders, and follow-up notifications are sent if needed

Use Case UC4: Save Favorite Route

- The Passenger selects a route within the ViaVan service that they frequently travel or are interested in. The system gives a favorite status to it and shows it in the favorite list. Passengers can quickly access their favorite route window afterward.

Use Case UC5: View Van Schedule

- The Passenger requests to view the schedule for ViaVan's vans. The system retrieves the schedule information and presents it to the Passenger in a table form.

Use Case UC7: Send feedback

- The Passenger sends feedback such as issues that they get when they are using ViaVan or when there is an issue with their driver. The feedback will be sent to customer services so that they can see the feedback and if it is a negative feedback, they can solve the issue later.

Use Case UC8: Payment

- Before the passenger completes their booking process. The system will show the amount that must be paid. Then the passenger must select payment methods such as Transfer with Paypal, Bank Account, Debit Card. And then the passenger will confirm their payment. After that the passenger must send a request to the payment service system in order to complete the confirmation process. After the payment, the system will display a result to the passenger that the payment has finished.

Use case Description (Fully-dressed format)

Use Case UC2: Track Van Location

Level: Passenger-level goal

Primary Actor: Passenger, GPS system

Preconditions:

- Passenger has access to the tracking feature.
- The van is equipped with a GPS tracking device that is active and transmitting data.

Stakeholders and their Interests (tracking):

- Passenger: Wants to track the van's location to estimate arrival times and plan accordingly.
- GPS System: Aims to provide real-time location data to the Passenger accurately and reliably.
- Driver: Wants to ensure that the tracking system does not infringe on personal privacy beyond work requirements.
- Company: Seeks to enhance customer satisfaction and operational efficiency by offering transparent tracking services.

Main Success Scenario (tracking):

- Passenger open the tracking system or app
- Passenger selects the service or delivery they wish to track.
- GPS System displays the real-time location of the van on a map interface.
- Passengers can view the van's estimated time of arrival and current status.
- If necessary, the Passenger can make informed decisions based on the van's location, such as rescheduling their time or contacting the company for further information.

Extensions (or Alternatives):

a. If the GPS signal is lost or the van is not transmitting location data:

- System notifies the Passenger of the issue and provides the last known location.
- Passengers may contact customer support if further clarification is needed.

b. Passenger wants to receive notifications about the van's movement:

- Passenger sets up alerts for changes in the van's status (e.g., when it's nearing the destination).
- System sends real-time notifications to the Passenger as per the set preferences.

c. If the Passenger cannot find the van's location due to system error:

- Passenger contacts customer support for assistance.
- Customer support uses internal systems to assist the Passenger and resolve the issue.

Use Case UC6: Booking-Van

Level: Passenger-level goal

Primary Actor: Passenger

Preconditions:

• Passenger is identified and authenticated in the booking system.

Stakeholders and their Interests:

- Passenger: Wants a smooth and efficient booking process with accurate information provided.
- System: Aims to facilitate the booking process seamlessly, ensuring accurate recording of passenger details and preferences.
- Company: Seeks to satisfy passenger needs and preferences, ensuring a positive experience and securing revenue.
- Government Agencies: May have regulatory interests related to passenger data and safety compliance.

Main Success Scenario:

- Passenger accesses the booking system to make a reservation.
- Passenger enters boarding point, destination
- System suggests the routes.
- Passenger selects the route.
- System checks availability and presents seat options.
- Passenger selects seats.
- System records booking details and generates a confirmation.
- Passenger receives confirmation and completes the booking process.

Extensions (or Alternatives):

a. If the system fails at any point:

- Passengers may need to restart the booking process or contact customer support for assistance.

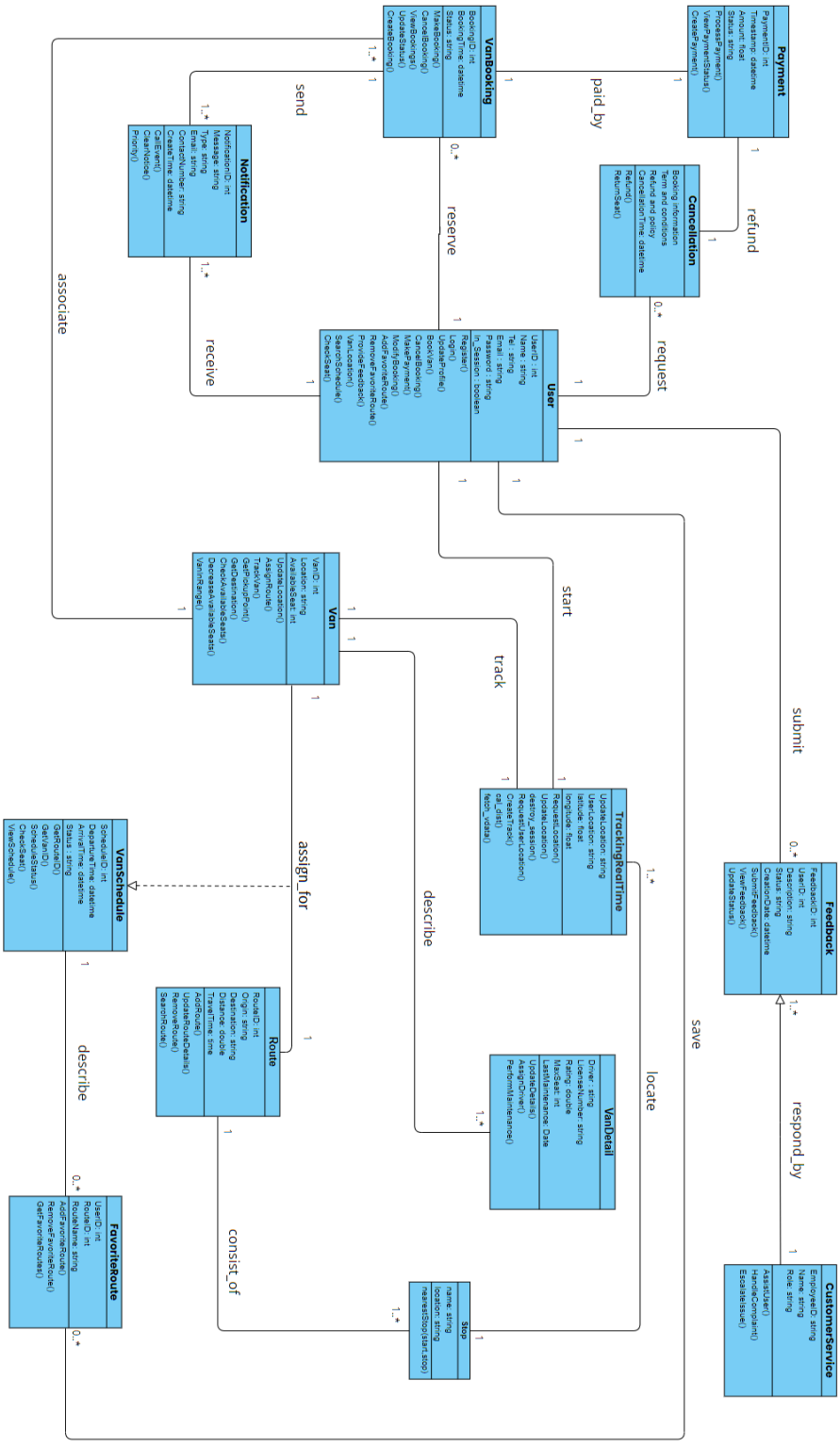
b. Passenger wants to modify or cancel a booking:

- Passenger accesses their booking through the app.
- Passenger selects the modification or cancellation option and follows the prompts.
- System updates the booking accordingly and notifies the Passenger.

c. If there are issues with overbooking:

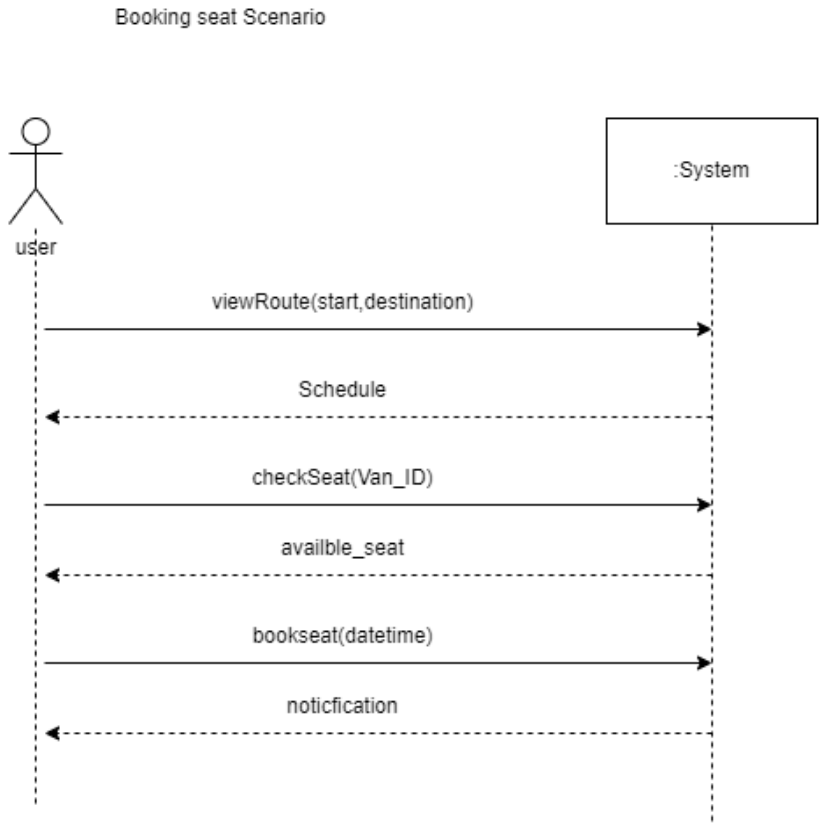
- System detects that the selected van is fully booked.
- System presents the Passenger with alternative van options or offers compensation for voluntary rescheduling.
- Passenger selects a suitable alternative or declines and seeks alternative transportation arrangements.

Class Diagram

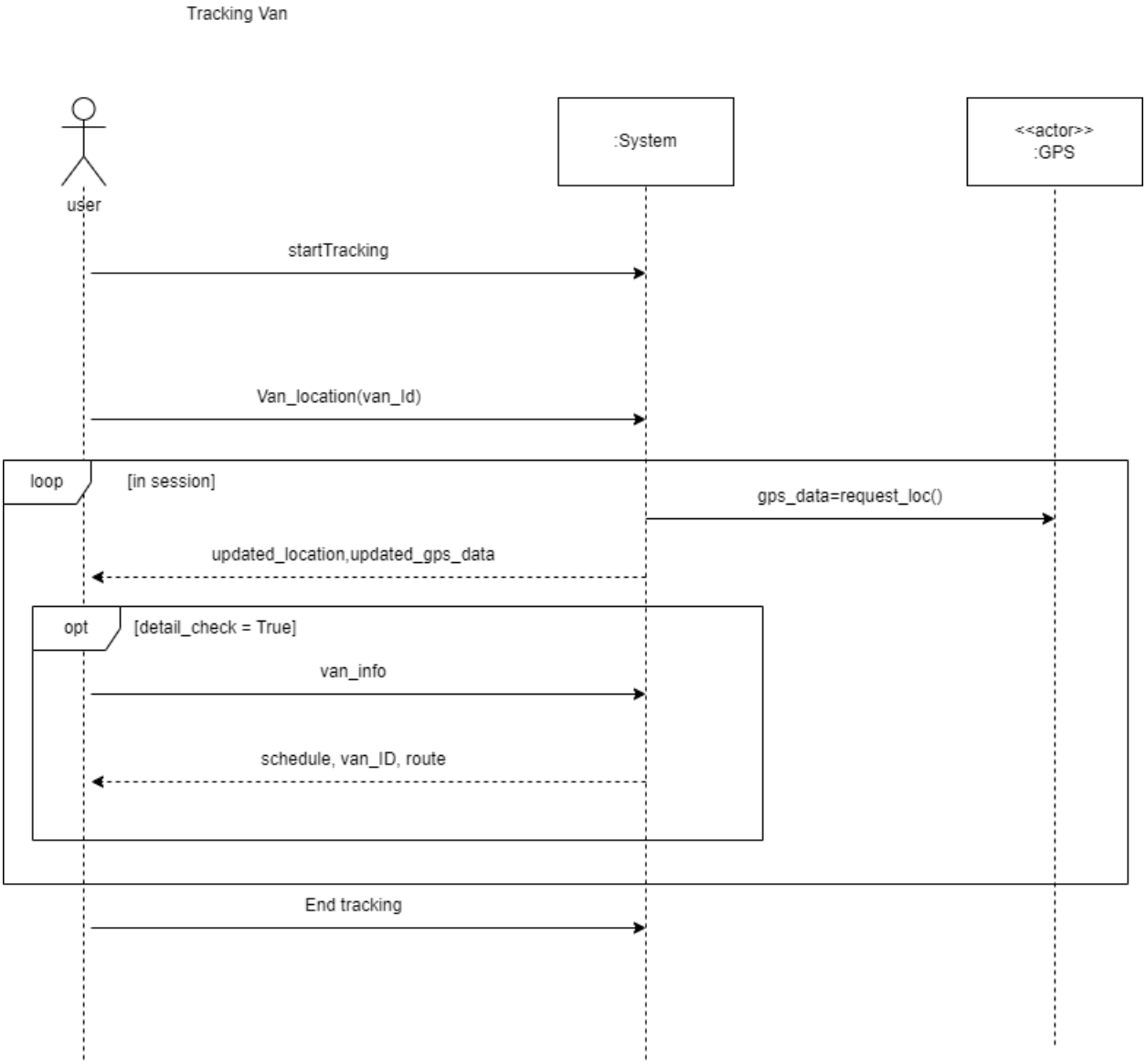


System Sequence Diagram (SSDs)

Booking Seat scenario



Tracking Van scenario



Operation contract

Operation Contract: BookSeat

Operation: viewRoute(start: String, destination: String)

Cross References: Use Case: View van Schedule
Preconditions:

1. The User must be registered and logged into the ViaVan app (existence of a User instance)
2. There must be at least one Route that matches the start and destination criteria (existence of Route instances).

Postconditions:

1. The system retrieves Route instances of sh list of ScheduleID (information retrieval).

Operation: checkSeat(van: Van)

Cross References: Use Case: booking van
Class Diagram: Involves the Van class.
Preconditions:

1. A Van instance corresponding to the Van_ID must exist (existence of a Van instance).
2. The Van instance must be operational and assigned to a Route (association formed between Van and Route).

Postconditions:

1. The system retrieves the AvailableSeat attribute of the Van instance (information retrieval).

Operation: bookVan(van van, date datetime)

Cross References: Use Case: booking van
Preconditions:

1. The User [In_session = True] (existence of a User instance).
2. A Van must exist for the selected Route with a status indicating it is operating on that route (existence of Van and Route instances, and an association between them).
3. The Van must have an available seat [available_seat > 0].

Postconditions:

1. A new instance of VanBooking is created (instance creation).
2. The Booking attribute in the VanBooking class is set to datetime (attribute modification).
3. The User is associated with the new VanBooking (association formed).
4. The status attribute of the VanBooking instance is set to "confirmed" (attribute modification).
5. A Notification instance is created to confirm the booking to the User (instance creation).
6. The AvailableCapacity attribute of the Van may be decremented, reflecting one less available seat (attribute modification)

Operation Contract: startTracking

Operation: startTracking()
Cross References: Use Case: Track Van Location

Preconditions:
1. The User must be registered and logged into the AppViaVan [In_session = True].
2. The Van must be registered in the system with a valid VanID.
3. The GPS system is operational and can provide location updates.

Postconditions:
1. A trackingRealtime instance ts is created (instance creation).
2. ts associated with the specific VanID for each within 500m (association formed).

Operation: Van_location(int van_id)
Cross References: Use Case: Track Van Location
Preconditions:
1. The startTracking operation has been successfully invoked.
2. The System has a valid VanID to query.

Postconditions:
1. The updated location became the Van.location (attribute modification).

Operation: van_info()
Cross References: Use Case: Track Van Location
Preconditions:
1. The User has requested additional details on the van (triggered by detail_check being true).

Postconditions:
1. The System retrieves instance Vinf (VanDetail) (information retrieval).

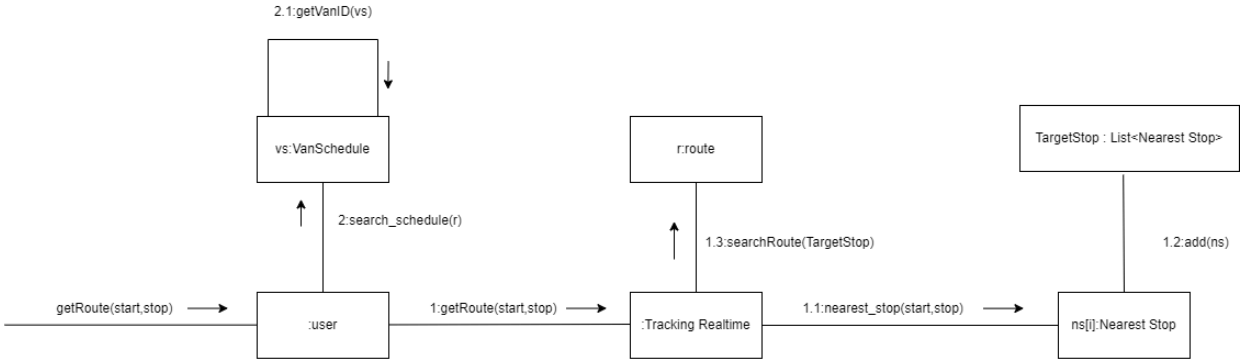
Operation: End tracking()
Cross References: Use Case: Track Van Location
Preconditions:
1. The User has been tracking a van's location.
2. The User decides to stop tracking the van.

Postconditions:
1. Each ts* associated with the VanID was break (association break).
2. ts is terminated (instance destruction).

Object-interaction diagram

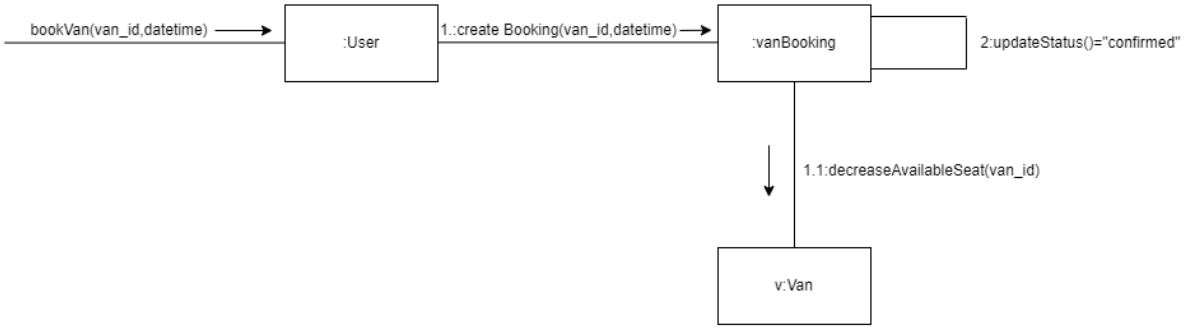
Operation Contract: BookSeat

Operation: viewRoute(start: String, destination: String)



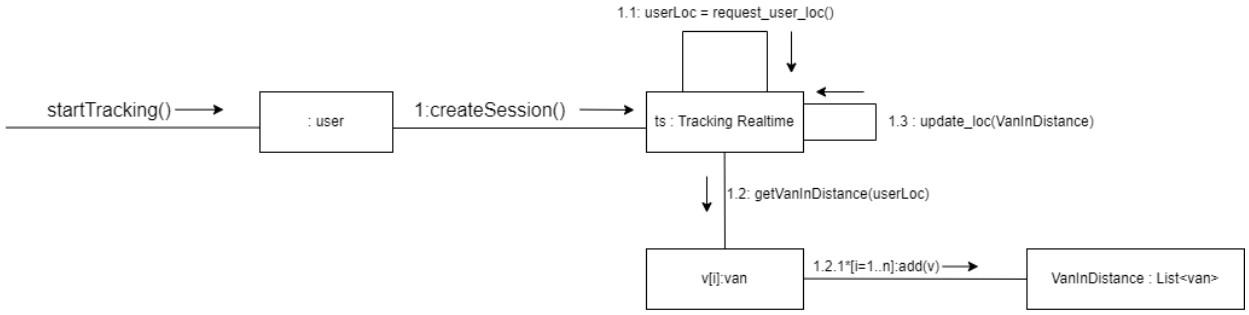
Operation: checkSeat(van: Van)

Operation: bookVan(van van, date datetime)

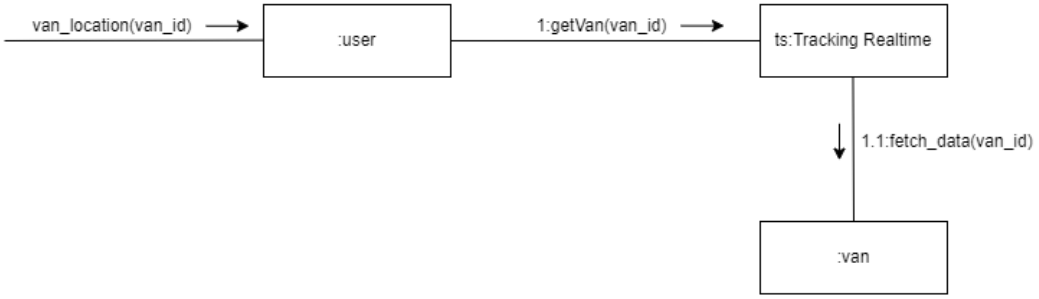


Operation Contract: startTracking

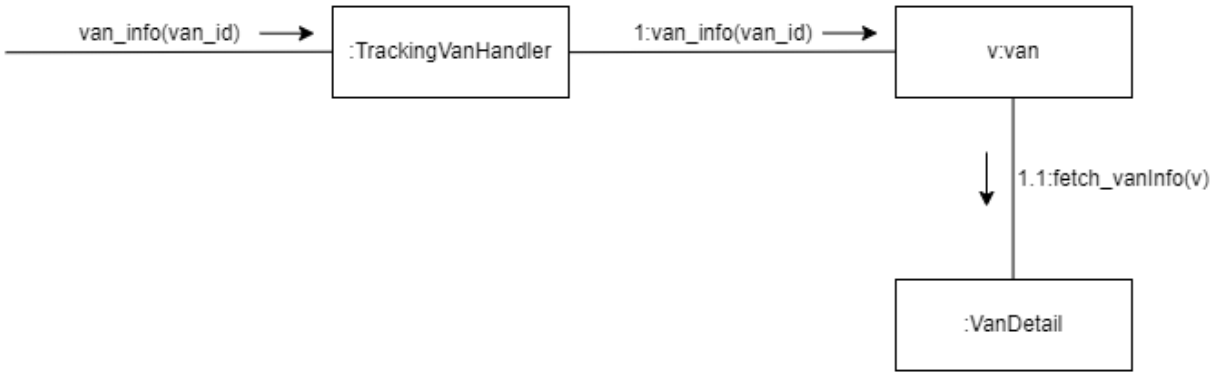
Operation: startTracking()



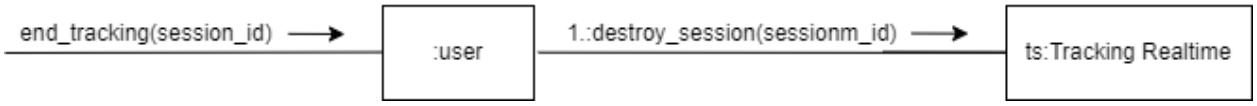
Operation: Van_location(int van_id)



Operation: van_info()



Operation: End tracking()



Object Constraint Language (OCL)

1. **The start and destination locations of a route cannot be the same.**
   Context class: Route
   Invariants: self.startLocation <> self.endLocation

2. **Feedback must be processed within 48 hours.**
   context class: Feedback
   Invariants: Date.today() - self.creationDate <= 2

3. **Booking must be associated with a valid user and van, and must have a status.**
   context class: VanBooking
   invariants: self.user->notEmpty() and
   self.van->notEmpty() and
   self.status->includes('confirmed' | 'canceled' | 'pending')

4. **Every notification must be associated with at least one user and contain message content.**
   context class: Notification
   invariants: self.user->notEmpty() and self.message.size() > 0

5. **Each schedule entry must specify a valid day of the week and times.**
   context class: VanSchedule
   invariants: self.dayOfWeek->notEmpty() and
   self.startTime->notEmpty() and
   self.endTime->notEmpty() and
   self.startTime < self.endTime