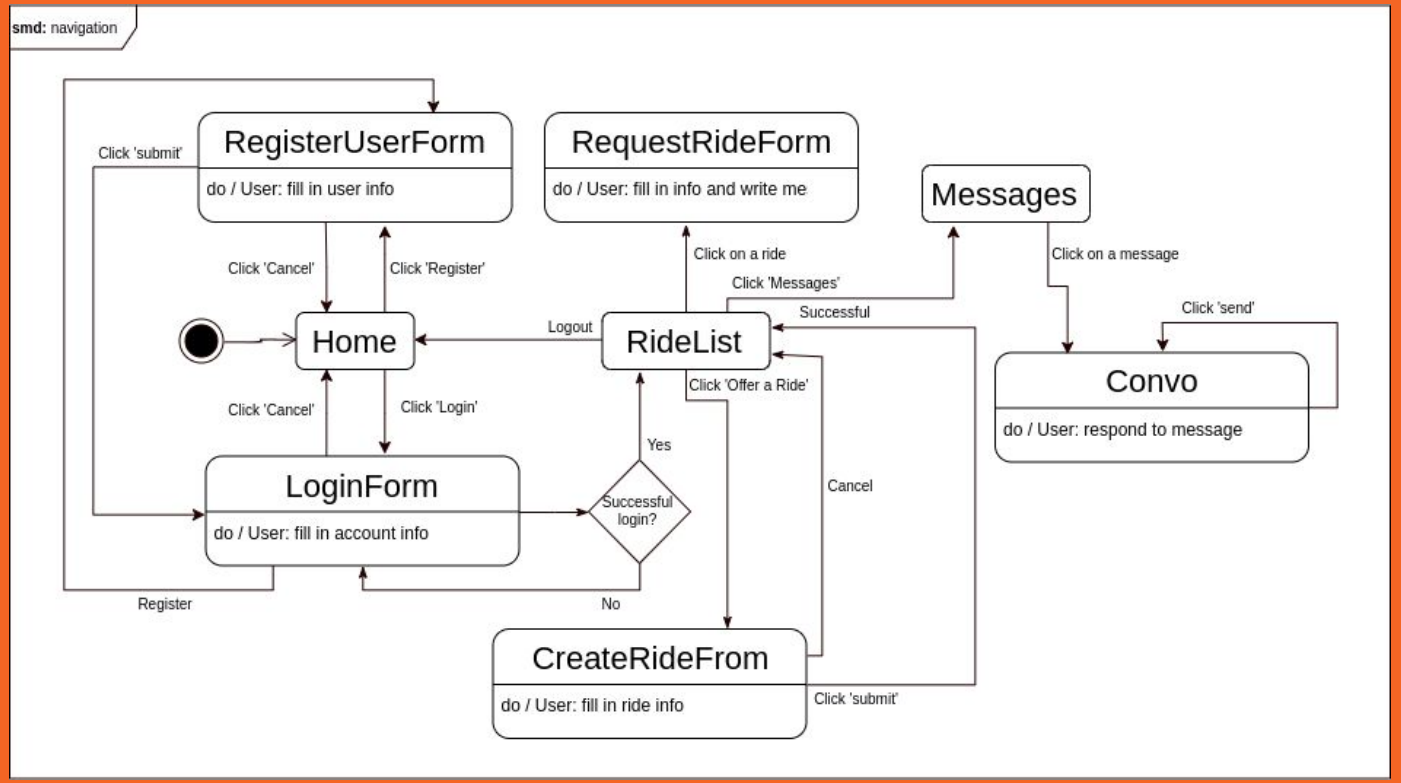
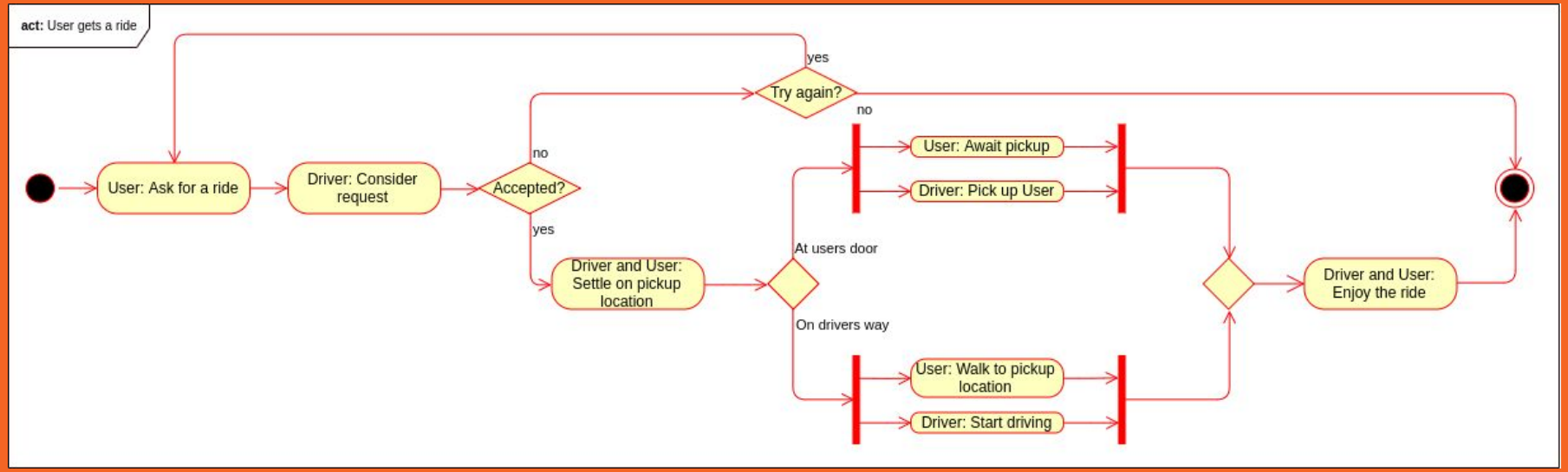
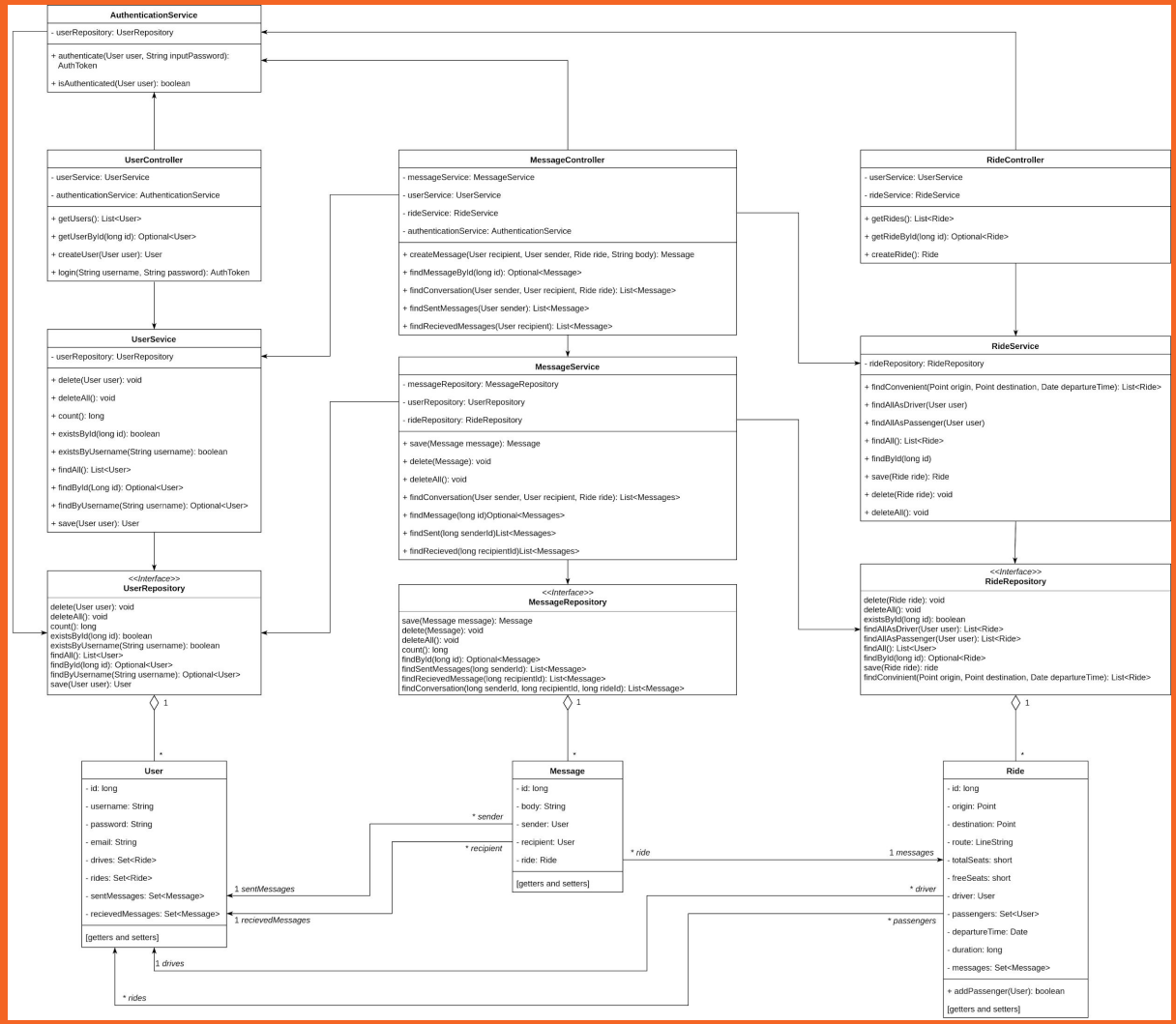
Hop on

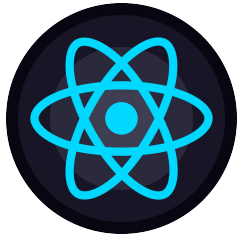
A ride sharing application

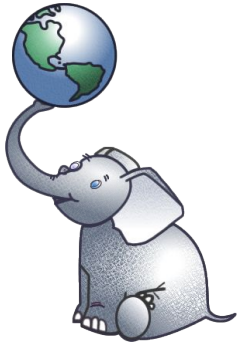
Gunnar & Jón


smd: navigation
RegisterUserForm
do / User: fill in user info
RequestRideForm
do / User: fill in info and write me
Messages
Click 'submit'
Click 'Cancel'
Click 'Register'
Click on a ride
Click 'Messages'
Click on a message
Successful
Click 'send'
Home
Logout
RideList
Convo
do / User: respond to message
Click 'Cancel'
Click 'Login'
Click 'Offer a Ride'
Yes
LoginForm
do / User: fill in account info
Successful login?
Cancel
Register
No
CreateRideFrom
do / User: fill in ride info
Click 'submit'

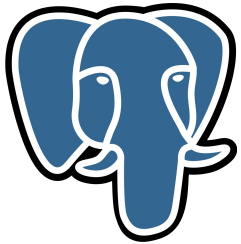

act: User gets a ride
User: Ask for a ride
Driver: Consider request
Accepted?
no
yes
Try again?
yes
no
Driver and User: Settle on pickup location
At users door
On drivers way
User: Await pickup
Driver: Pick up User
User: Walk to pickup location
Driver: Start driving
Driver and User: Enjoy the ride


AuthenticationService
- userRepository: UserRepository
+ authenticate(User user, String inputPassword): AuthToken
+ isAuthenticated(User user): boolean
UserController
- userService: UserService
- authenticationService: AuthenticationService
+ getUsers(): List<User>
+ getUserById(long id): Optional<User>
+ createUser(User user): User
+ login(String username, String password): AuthToken
MessageController
- messageService: MessageService
- userService: UserService
- rideService: RideService
- authenticationService: AuthenticationService
+ createMessage(User recipient, User sender, Ride ride, String body): Message
+ findMessageById(long id): Optional<Message>
+ findConversation(User sender, User recipient, Ride ride): List<Message>
+ findSentMessages(User sender): List<Message>
+ findRecievedMessages(User recipient): List<Message>
RideController
- userService: UserService
- rideService: RideService
+ getRides(): List<Ride>
+ getRideById(long id): Optional<Ride>
+ createRide(): Ride
UserSevice
- userRepository: UserRepository
+ delete(User user): void
+ deleteAll(): void
+ count(): long
+ existsById(long id): boolean
+ existsByUsername(String username): boolean
+ findAll(): List<User>
+ findById(Long id): Optional<User>
+ findByUsername(String username): Optional<User>
+ save(User user): User
MessageService
- messageRepository: MessageRepository
- userRepository: UserRepository
- rideRepository: RideRepository
+ save(Message message): Message
+ delete(Message): void
+ deleteAll(): void
+ findConversation(User sender, User recipient, Ride ride): List<Messages>
+ findMessage(long id)Optional<Messages>
+ findSent(long senderId)List<Messages>
+ findRecieved(long recipientId)List<Messages>
RideService
- rideRepository: RideRepository
+ findConvenient(Point origin, Point destination, Date departureTime): List<Ride>
+ findAllAsDriver(User user)
+ findAllAsPassenger(User user)
+ findAll(): List<Ride>
+ findById(long id)
+ save(Ride ride): Ride
+ delete(Ride ride): void
+ deleteAll(): void
<<Interface>>
UserRepository
delete(User user): void
deleteAll(): void
count(): long
existsById(long id): boolean
existsByUsername(String username): boolean
findAll(): List<User>
findById(long id): Optional<User>
findByUsername(String username): Optional<User>
save(User user): User
<<Interface>>
MessageRepository
save(Message message): Message
delete(Message): void
deleteAll(): void
count(): long
findById(long id): Optional<Message>
findSentMessages(long senderId): List<Message>
findRecievedMessage(long recipientId): List<Message>
findConversation(long senderId, long recipientId, long rideId): List<Message>
<<Interface>>
RideRepository
delete(Ride ride): void
deleteAll(): void
existsById(long id): boolean
findAllAsDriver(User user): List<Ride>
findAllAsPassenger(User user): List<Ride>
findAll(): List<User>
findById(long id): Optional<Ride>
save(Ride ride): ride
findConvinient(Point origin, Point destination, Date departureTime): List<Ride>
1
*
1
*
1
*
User
- id: long
- username: String
- password: String
- email: String
- drives: Set<Ride>
- rides: Set<Ride>
- sentMessages: Set<Message>
- recievedMessages: Set<Message>
[getters and setters]
Message
- id: long
- body: String
- sender: User
- recipient: User
- ride: Ride
[getters and setters]
Ride
- id: long
- origin: Point
- destination: Point
- route: LineString
- totalSeats: short
- freeSeats: short
- driver: User
- passengers: Set<User>
- departureTime: Date
- duration: long
- messages: Set<Message>
+ addPassenger(User): boolean
[getters and setters]
* sender
* recipient
* ride
1 messages
1 sentMessages
1 recievedMessages
* driver
* passengers
1 drives
* rides





openroute service



Implementation and Tech Stack

Backend

- Java Spring REST API
- PostgreSQL
- PostGIS psql extension for geospatial data
- Hosted on Heroku

Frontend

- Reactjs
- Leafletjs
- React-leaflet

Other

- Uses the Open Route Service API extensively
  - Route directions
  - Geocoding and naming
- APIS petrol endpoint
  - To calculate fuel costs
- Client-side rendering
  - Leaflet does not support ssr
- JWT authentication scheme

Retrospect

- The project was fairly consistent with the Design model
  - Minor changes to the backend
    - Did not(could not) anticipate exactly how and what data the frontend would consume
  - Lots of unfamiliar technologies
    - Resulted in some inconsistencies and possibly antipatterns
    - Some things took longer than we intuitively thought
    - Took too long to figure out a good way to sync React with Leaflet
  - Authentication is implemented very differently than originally planned(apart from using JWT)
  - Frontend had no formal planning or modeling, but is consistent with the original vision

What could have gone better?

- Developing the backend went pretty much as expected
  - Could have been made more flexible in hindsight
  - The original plan was missing some features that made developing the frontend harder than it needed to be
  - Not enough work put into high quality error messages
- The frontend was difficult
  - Leaflet and React don't play super nicely together
  - The API was not flexible enough which added a lot of overhead with data fetching
    - Probably could have been alleviated somewhat by doing less with it on the backend
  - Probably would have been a good idea to adopt a state management solution, like Redux or Mobx
  - Should have used Typescript

Demo

Questions?