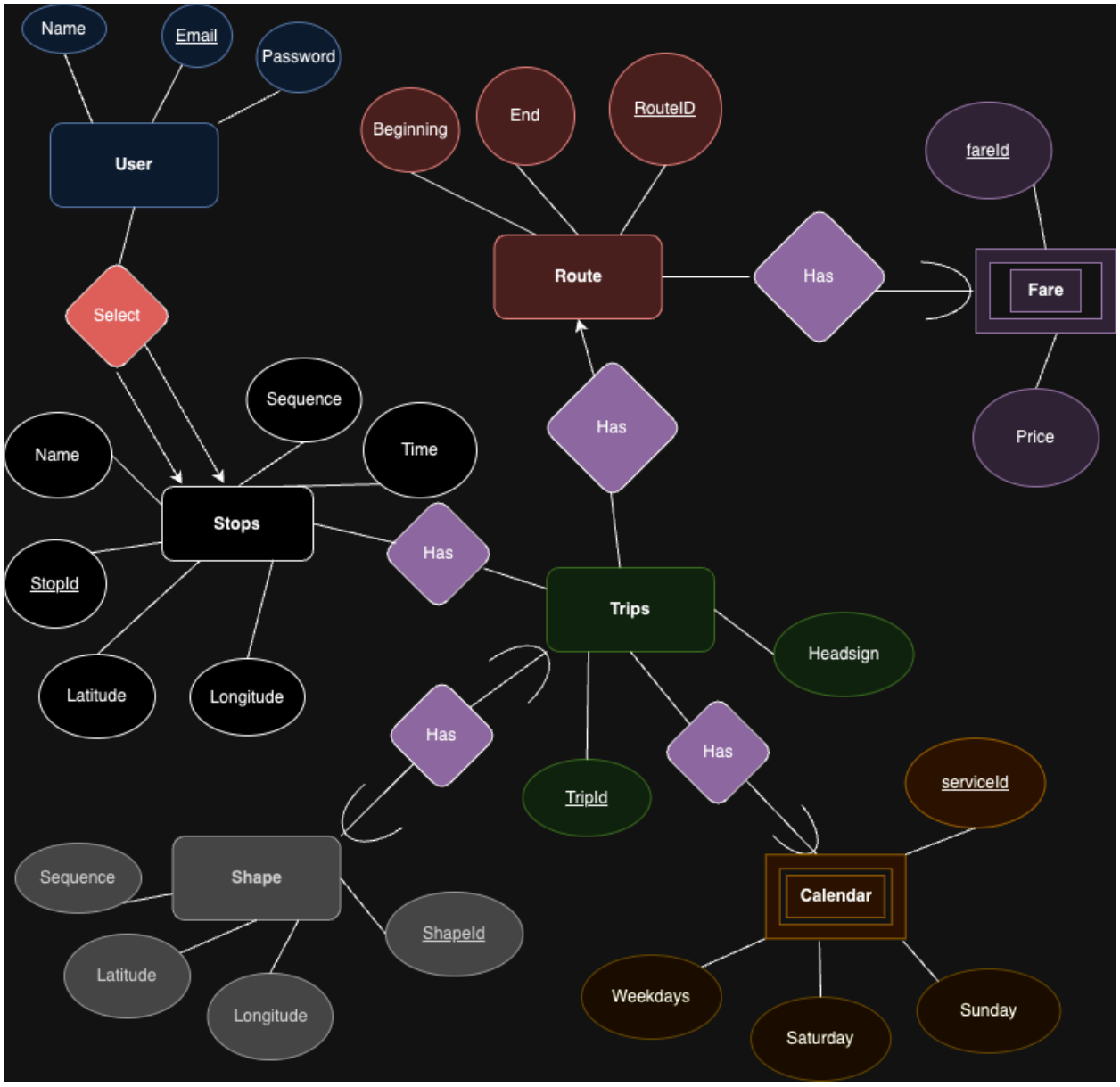
ER Diagram:



Assumptions and Descriptions

- Each user will have 3 attributes, which include their name, email, and password. Each user will be uniquely identified by his or her email address. We'll assume a user will select two stops (one for where they get on and one for where they get off).We put two arrows from the user selects stops because the user can pick at most one starting stop and at most one destination stop. However, each stop can be selected by multiple users. Each stop has 6 attributes: Name, StopId, Latitude, Longitude, Sequence, and Time. Stops are uniquely identified by StopId. Multiple stops can be used in a trip, and each trip can have multiple stops. Trips will be uniquely identified by TripId and will have another attribute called headsign. Each trip will have at max 1 specific shape and each shape can have at

max 1 trip. Each shape has three attributes: Sequence, Latitude, Longitude, and ShapeId. ShapeId is used to uniquely identify Shape. Trips will also have a calendar representing what days they operate. Since the calendar depends on the trip, it is a weak entity. Calendar has four attributes: Weekdays, Saturday, Sunday, and ServiceId. ServiceId is used to uniquely identify the calendar. This is a composite primary key with tripId since serviceId depends on tripId but serviceId allows us to assess whether a trip runs on weekdays, saturdays, or sundays. Additionally, we'll assume that each trip will have 0 or 1 route, but each route can be used for multiple trips. Each route has a Fare. Each route has exactly one fare since based on the data each specific route id seems to show up only once in the Fare table. However each specific fare seems to have multiple route ids. Thus each Fare can have multiple routes. Fare will be uniquely identified by Fare Id and will have a price attribute. The fareId is also underlined in our diagram because it is a composite primary key with routeId. FareId is used to access the price for a fare but fareId is also dependent on routeId.

Relational Schema:

User (Email:VARCHAR(32) [PK], Password: VARCHAR(32), Name: VARCHAR(32), StopId1[FK to Stops.StopId], StopId2 [FK to Stops.StopId])

Route (RouteId: INT[PK], Beginning: VARCHAR(32), End: VARCHAR(32))

Stops (StopId: INT[PK], Longitude: REAL, Latitude: REAL, Name: VARCHAR(32), Sequence: , Time: time, TripId: INT [FK to Trips.TripId])

Fare (fareId: INT [PK], RouteId: INT [PK], Price: REAL)

Trips (TripID: INT[PK], Headsign: VARCHAR(32), ShapeId:INT[FK to Shape.ShapeId])

Shape (ShapeId: INT[PK], Longitude: REAL, Latitude: REAL, Sequence: INT, TripId: INT[FK to Trips.TripId])

Calendar (TripId: INT [PK], serviceId: VARCHAR(32) [PK], Weekdays: VARCHAR(32), Saturday: VARCHAR(32), Sunday: VARCHAR(32))

Normalization (3NF):

All dependencies:

fare.fareId → fare.price
route.routeId → fare.fareId
user.email → user.name
user.email → user.password
trips.tripId → shape.shapeId
route.routeId → route.beginning
route.routeId → route.end
trips.tripId → route.routeId (one route can have one or two trips - many to one)
trips.tripId → trip.headsign
trips.tripId → calendar.serviceId
calendar.serviceId → calendar.weekdays
calendar.serviceId → calendar.saturday
calendar.serviceId → calendar.sunday
trips.tripId → stops.stopId
stops.stopId → stops.sequence
stops.stopId → stops.time
stops.stopId → stops.name
stops.stopId → stops.latitude
stops.stopId → stops.longitude
shape.latitude → shape.shapeId (many latitudes to one shapeId)
shape.longitude → shape.shapeId (many longitudes to one shapeId)
shape.sequence → shape.shapeId (many sequences to one shapeId)
shape.shapeId → trips.tripId (one-to-one relationship)

After normalizing:

fare.fareId → fare.price
route.routeId → fare.fareId, route.beginning, route.end
user.email → user.name, user.password
trips.tripId → shape.shapeId, route.routeId, trip.headsign, calendar.serviceId, stops.stopId
calendar.serviceId → calendar.weekdays, calendar.saturday, calendar.sunday
stops.stopId → stops.sequence, stops.time, stops.name, stops.latitude, stops.longitude
shape.latitude → shape.shapeId
shape.longitude → shape.shapeId
shape.sequence → shape.shapeId
shape.shapeId → trips.tripId

R_1(fare.fareId, fare.price)
R_2(route.routeId, fare.fareId, route.beginning, route.end)

R_3(user.email, user.name, user.password)
R_4(trips.tripId, shape.shapeId, route.routeId, trip.headsign, calendar.serviceId, stops.stopId)
R_5(calendar.serviceId, calendar.weekdays, calendar.saturday, calendar.sunday)
R_6(stops.stopId, stops.sequence, stops.time, stops.name, stops.latitude, stops.longitude)
R_7(shape.latitude, shape.shapeId)
R_8(shape.longitude, shape.shapeId)
R_9(shape.sequence, shape.shapeId)
R_{10}(shape.shapeId, trips.tripId)
R_{11}(trips.tripId, user.email, shape.latitude, shape.longitude, shape.sequence) *candidate key*

We chose to use 3NF rather than BCNF because we wanted to preserve the dependencies in the data. Given the complexity of the data, we also decided to use 3NF because we are more familiar with it and it would make normalizing the data much easier.