Sprint 4
Apps and Timeframes

Apps, time limited pricing rules and special rates, on-meter price calculations

PM2 / TS Compatibility

Best case scenario: (on server)

1. `$ npm install pm2@latest -g`
2. `$ pm2 update`
3. `$ pm2 install typescript@2.6.2`

Worst case scenario solution: (locally)

1. Transpile .ts files locally
2. Push resulting .js files to the repository
3. Run project like normal on the server

Timeframes Proposal

Timeframes were added as a definition of time restrictions to rules and discounts by disabling and enabling hours of a repeating week schedule, between two timestamps

The hours of the week as binary

1. The hours that are active within a week can be stored as bits.
2. Each day would have 24 bits, so a week would have 24 * 7 = 168 bits.
3. 168 / 8 = 21 bytes of data storage would be required.
4. Scalability is limited by the ability to bulk update existing binary values, for example: when in the future half hours should be represented instead of hours. Updating from 21 bytes to 42 bytes.

Test 1: BinData

1. See the BSON spec for more information about BinData(number, string)
2. Type 0 would allow an arbitrary length of data to be stored.
3. Loopback stores encoded strings, making it hard to store and retrieve data.
4. When forcing loopback to store as buffer, the data cannot be read, and must manually be modified.

Inserting

Type 0 BinData is a generic BinData type that accepts generic formats

```
db.Timeframe.insert({

   startDate: new Date(2018, 4, 7),
   endDate: new Date(2019, 4, 7),
   weekSchedule: BinData(0,

     “00110100011001101100001 1
      01101011001100001011110 0
      10101010111010001111100 0
      11111001111101110010000 1
      10100000010111011001001 0
      11001000000100001010110 1
      01011110100000101001110”

   )
})
```

Test 2: boolean[]

1. A boolean[] can be queried by indexes, which makes it easy to find results where the indexes have a truthy value.
2. The query however must be constructed dynamically by adding clauses for each index. Errors may be thrown when indexes don’t match.
3. This solutions makes it hard to check more than one index, having to build complex queries in advance.

Inserting

An array of booleans can be quite verbose

```
db.Timeframe.insert({

   startDate: new Date(2018, 4, 7),
   endDate: new Date(2019, 4, 7),
   weekSchedule: [
      true,
      false,
      true,
      ...      // 162 more
      false,
      false,
      false
   ]

})
```

Querying

The resulting query matching values verbosity increases depending on amount of indexes checked

```
db.Discount.find({

  timeframes: {

    weekSchedule.3: true,
    weekSchedule.16: true,
    ...
    weekSchedule.128: true,
    weekSchedule.129: true,

  }

})
```

Test 3: string

1. A string is a very flexible datatype.
2. Using a regex in a query makes checking multiple bits in the string relatively easy, and enables different values next to 0 and 1.
3. It also makes querying the data really stable, as the query will silently fail if the content of the data is not of expected length or value.
4. Performance is not an issue if the regex column is indexed, and when prefix expressions (/^/) are used: docs.mongodb.com/manual/reference/operator/query/regex/#index-use
5. Another advantage is freedom and scalability. If multiple values, or ranges need to be matched, a simple regex modification is sufficient.

Inserting

1. startDate and endDate: absolute boundaries of the schedule
2. weekSchedule: every 24 bits represent 24 hours of a day of the week

```
db.Timeframe.insert({

   startDate: new Date(2018, 4, 7),
   endDate: new Date(2019, 4, 7),

   weekSchedule:

    “001101000110011011000011  //m
     011010110011000010111100  //t
     101010101110100011111000  //w
     111110011111011100100001  //t
     101000000101110111100100  //f
     110010000010000010101101  //s
     010111101000000101001110” //s

})
```

Querying

Patterns matching fields support indexing, improving performance

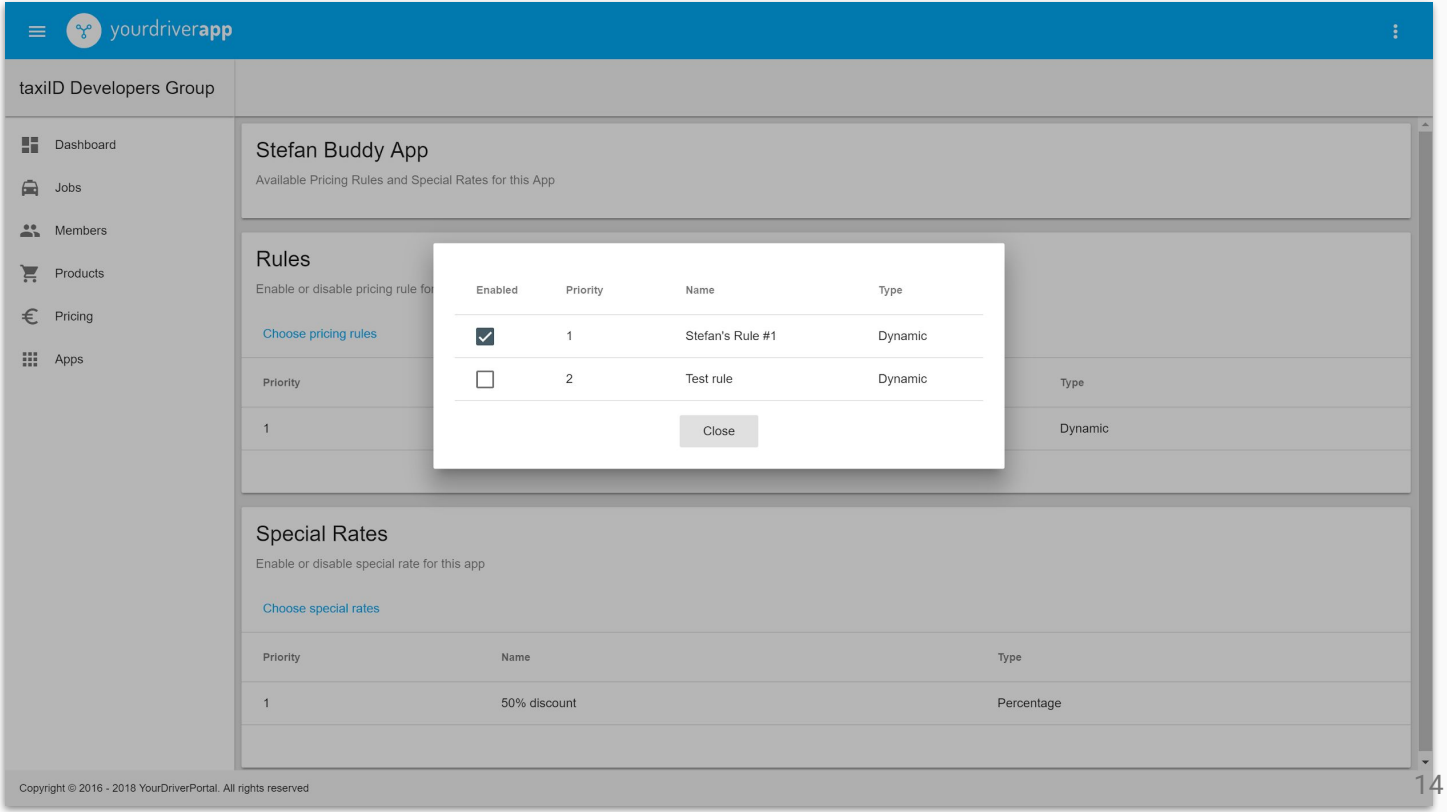
```
// First bit is 1
Db.Discount.find({

   "timeframes.weekSchedule": {
     $regex: /^1/
   }

})

// 2 4 6 8 10 12 15 bits are 1
/^.1.1.1.1.1.1..1/

// 128 129 and 131 bits are 1
/^.{127}11.1/
```

Views

The following slides show the views that were added during this sprint

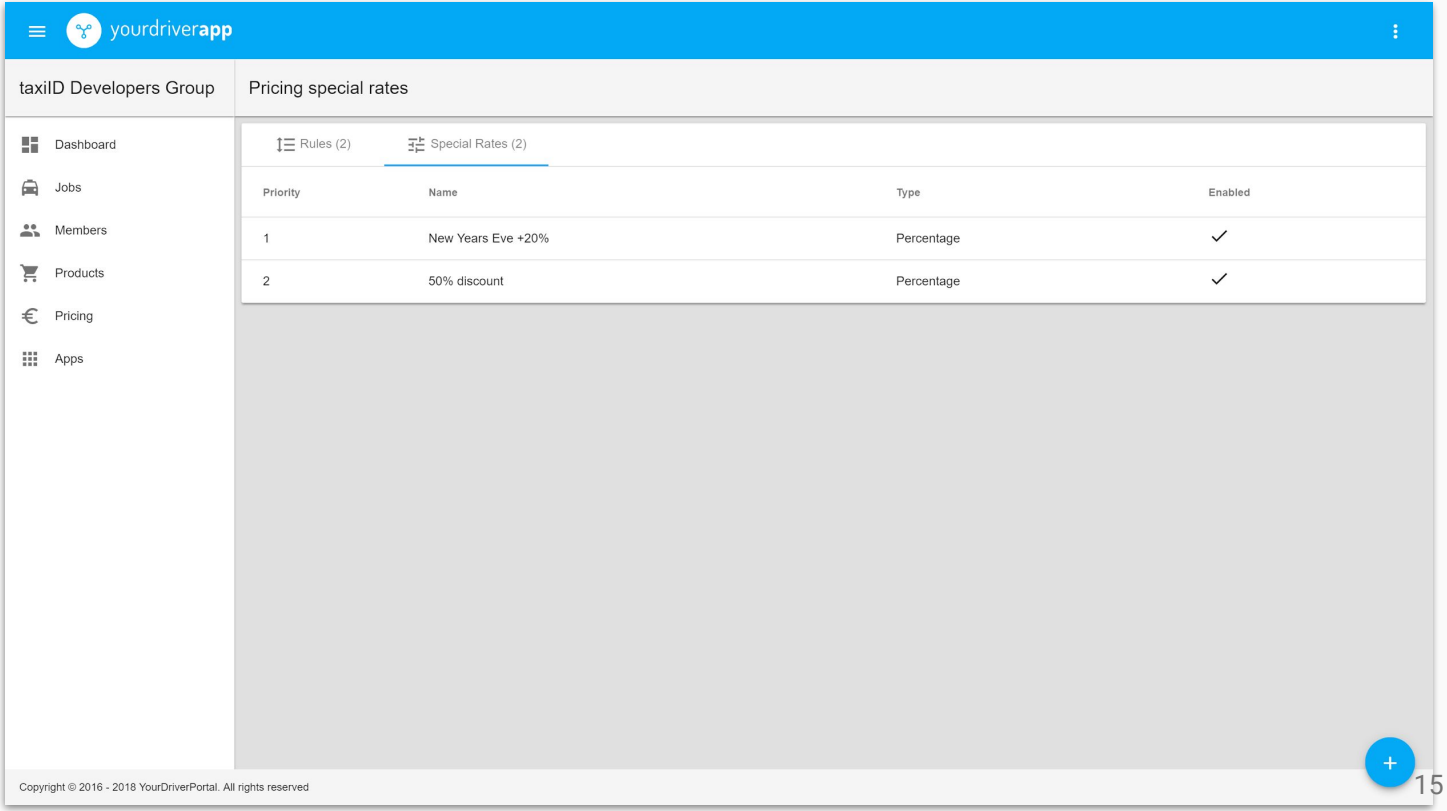
Apps, Enabled Rules and Special Rates

A modal opens up when ‘Change x’ is clicked, showing a table that allows users to add or remove pricing rules and / or special rates to the App



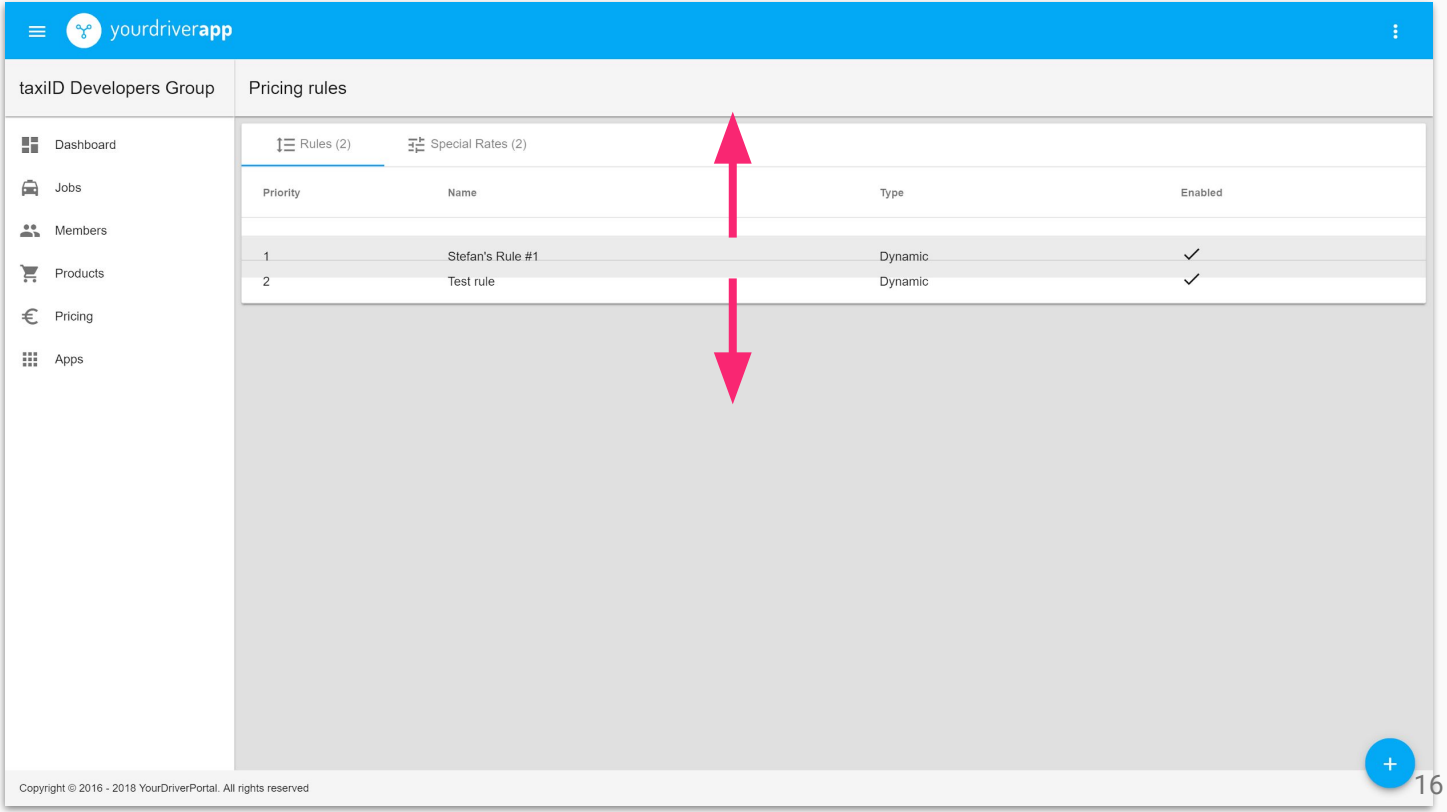
Pricing Rules / Special Rates

The Pricing tab now shows two tabs. Both having a table in which the rows can be sorted, using the exact same logic



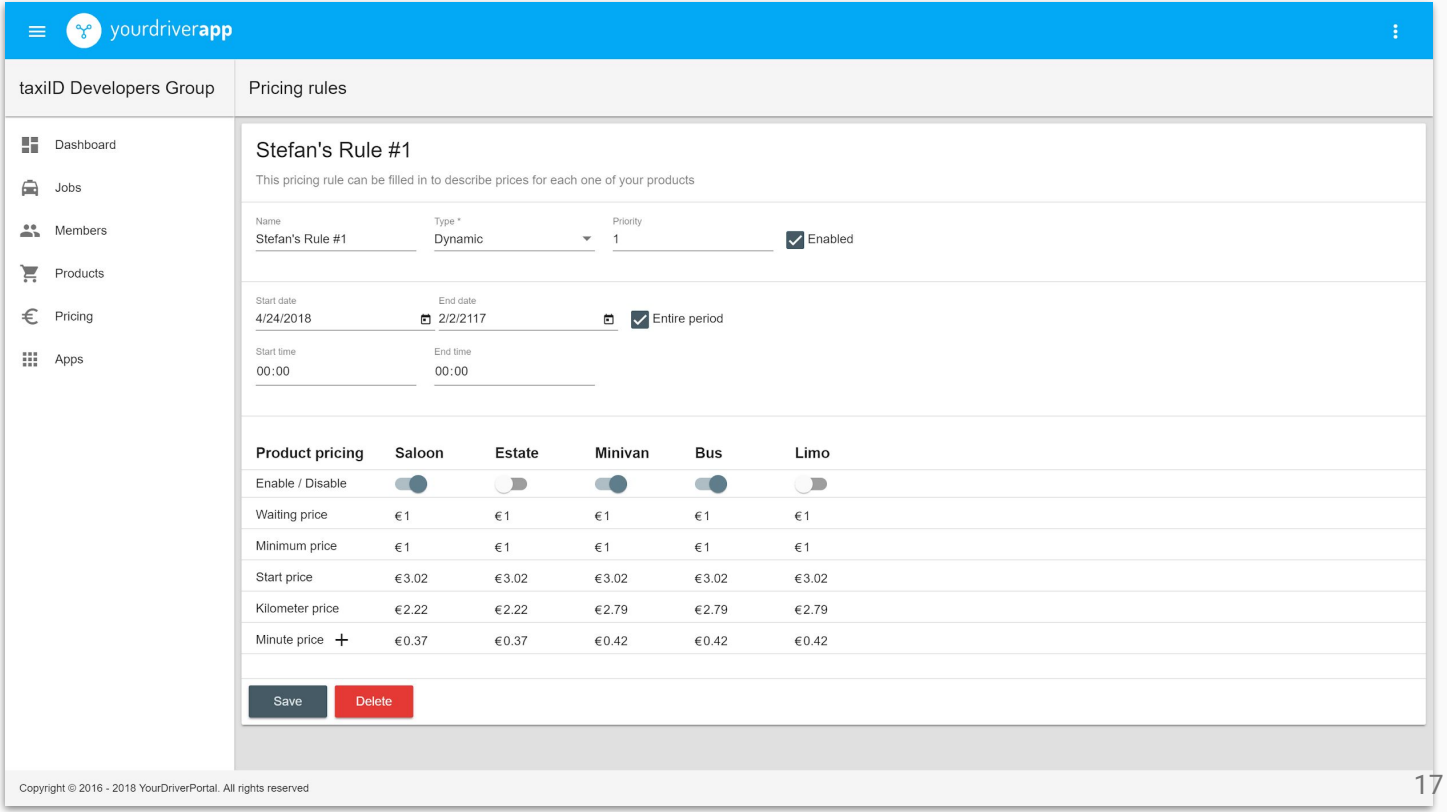
Dragging Pricing Rules / Special Rates

Automatically sorts stuff in the backend when an entitiy is picked up and dropped at some location



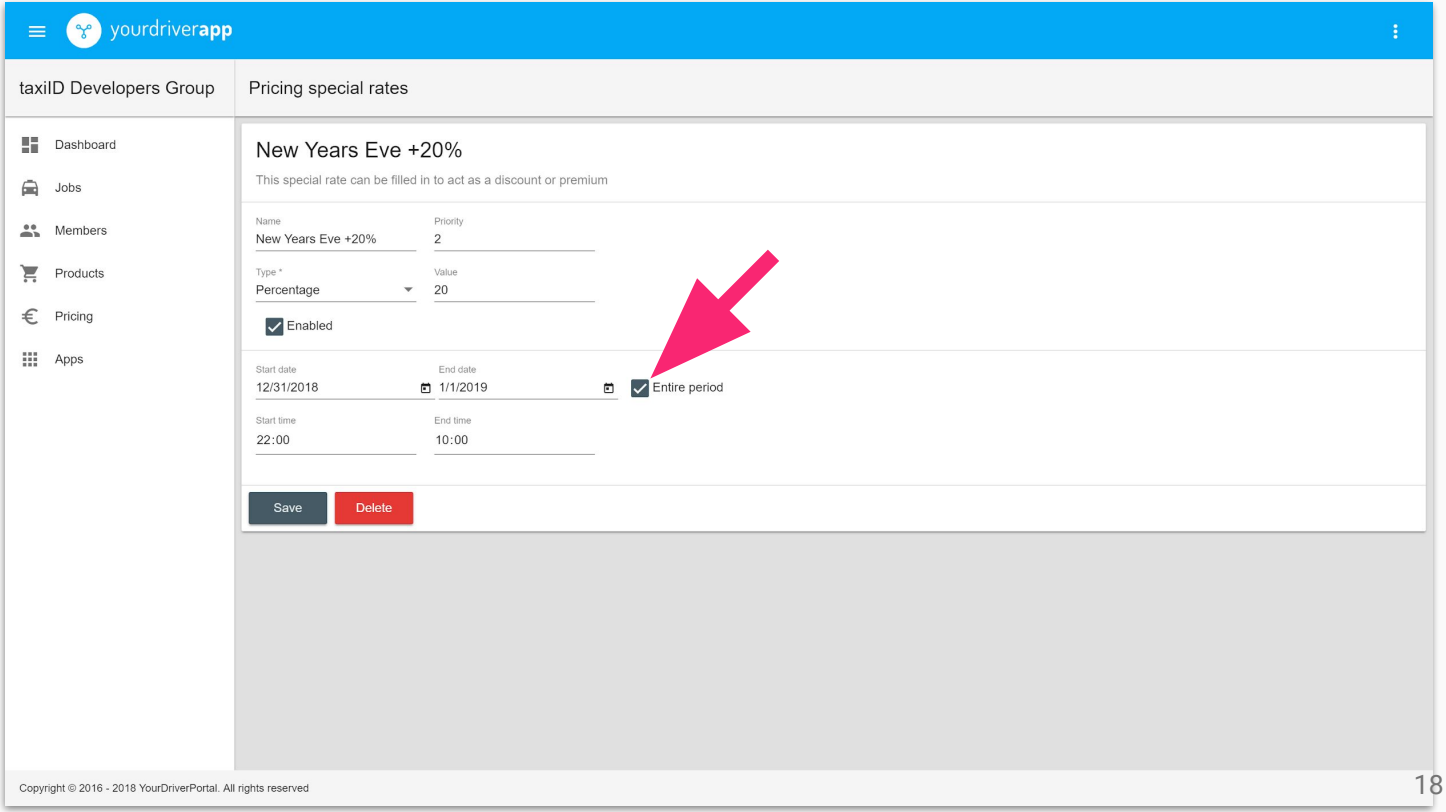
Pricing Rule

A timeframe is added to the pricing rule as a component



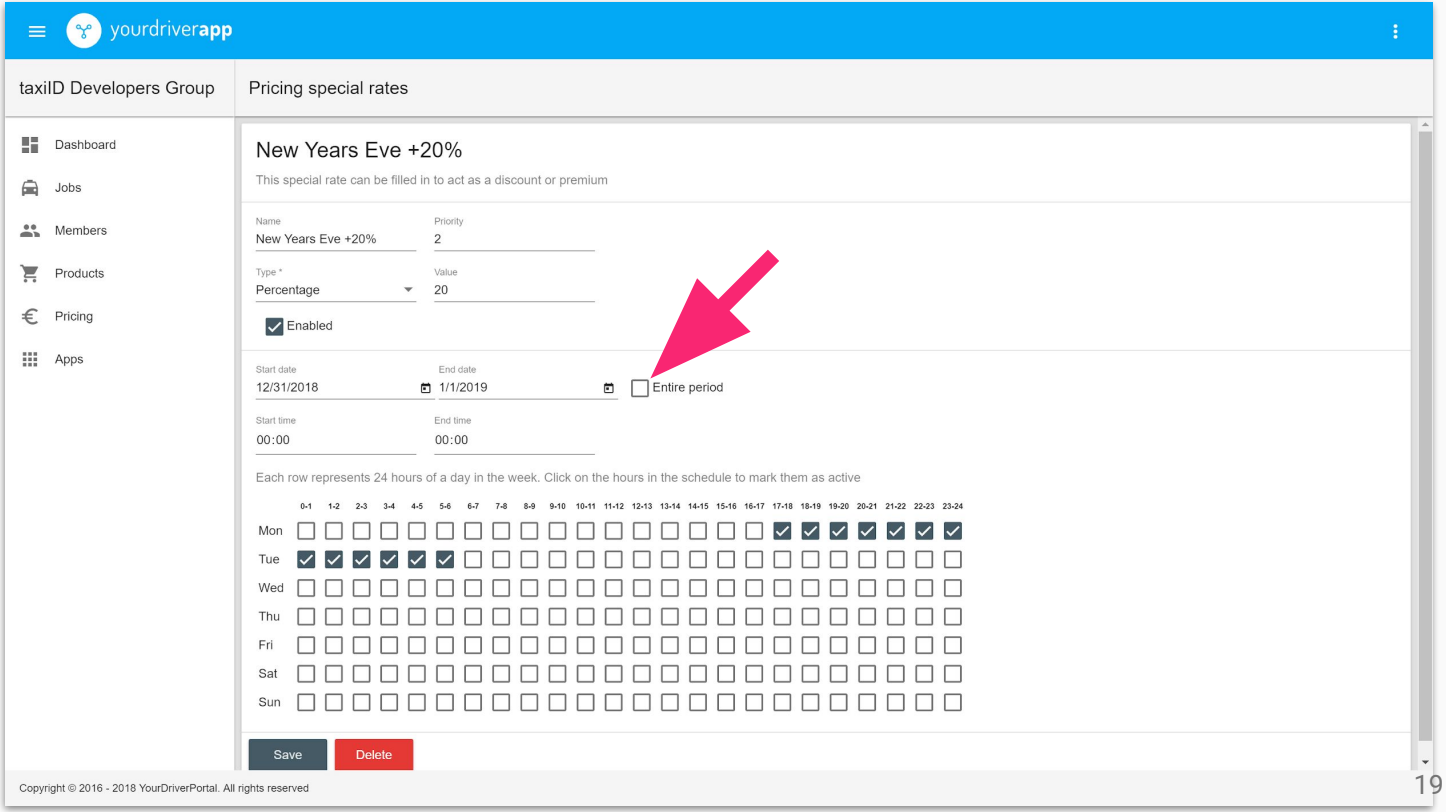
Special Rate

The Special Rate page is added, having the same timeframe component as the Pricing Rule page



Special Rate

Alternatively, when the ‘Entire Period’ checkbox is unchecked, the hour selector for the week is shown, allowing the user to customize every single hour of the week, from start till end date



Data Management

1. When a new pricing rule or discount is created, its priority will be 1, other element priorities get incremented
2. When an element is deleted, all elements with larger priority get decremented
3. When an element is modified, all other elements are modified so that the priority order *1* … *n* is maintained
4. A priority that is given, higher than *n* is capped at *n*
5. A priority lower than *1* is defaulted to *1*
6. A timeframe is added by default

Reflection - Must Haves

1. Thresholds that are incrementally bigger should be added to each pricing of a rule if the add button is pressed
2. Thresholds should be deletable with a simple button click
3. Display fixed special ratings and fixed threshold prices in € instead of cents, display percentages with the % symbol
4. Pricing Rules are stored in the database before save is pressed for technical simplicity. This should not be the case ideally

Reflection - Could Haves

1. Show warning if user has modified form fields, and tries to leave the page
2. Remove isEnabled flag for pricing rules and special rates, as they have to be enabled in the Apps view as well