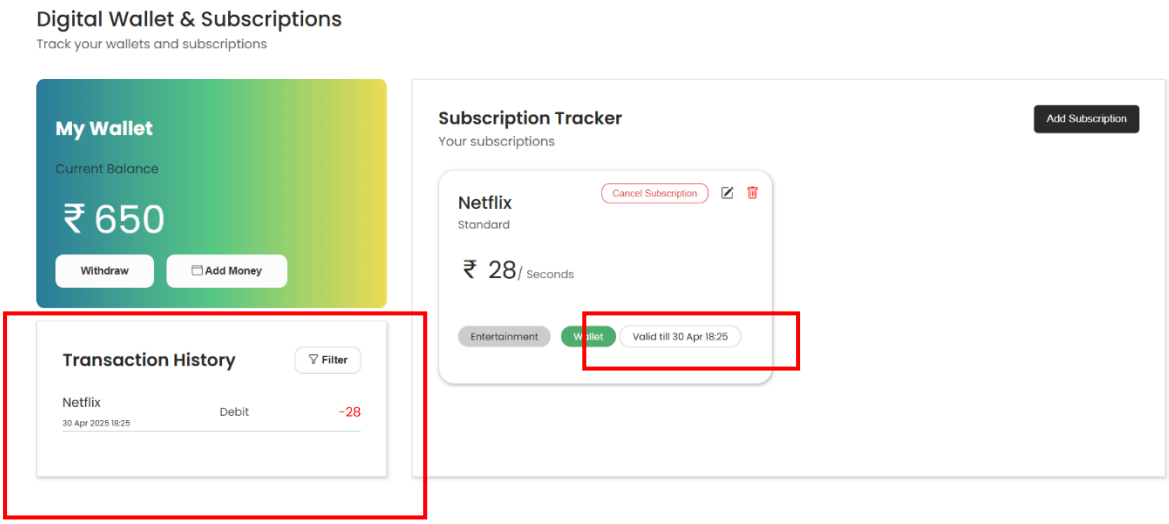
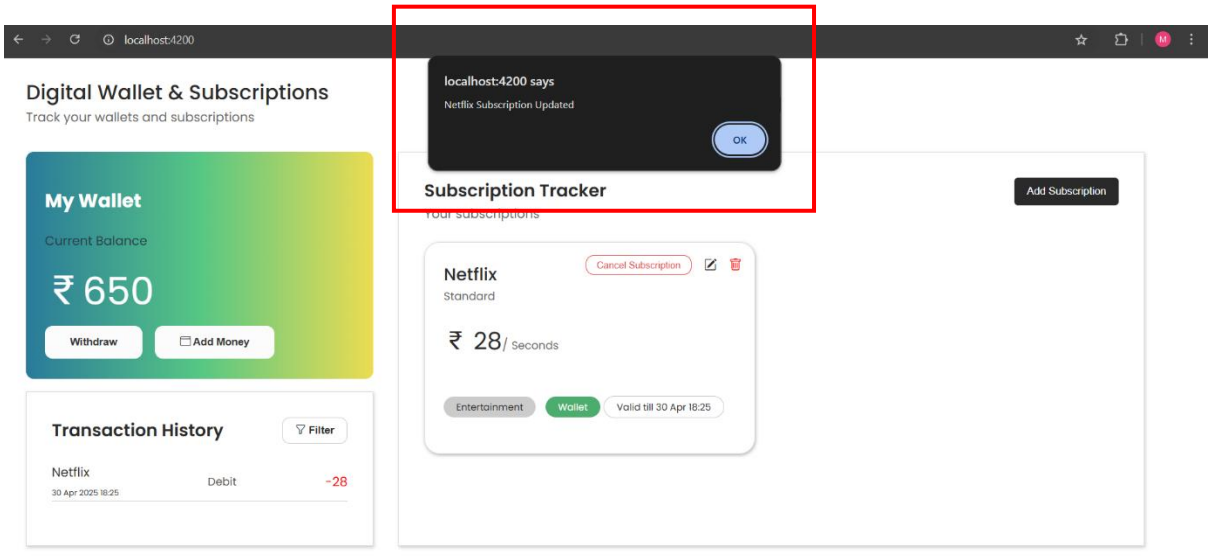
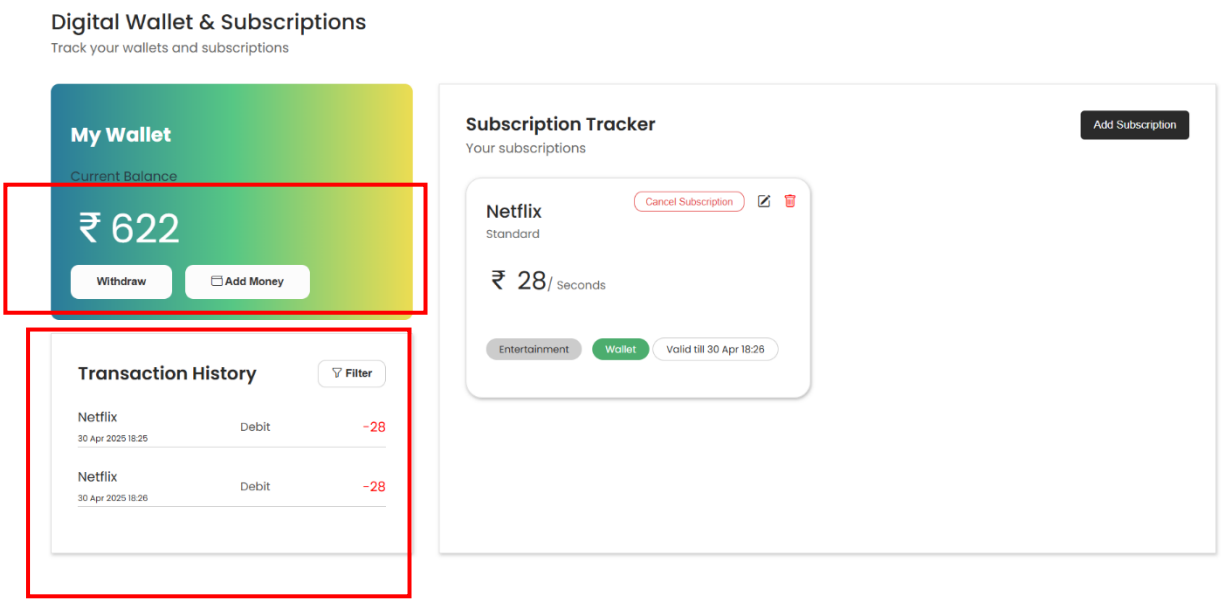
OUTPUT:

Task 9: Change the billing cycle to Seconds, Minutes and trigger subscription deduct amount from the wallet.







CODE:

Index.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>AssessmentApp</title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  {{content-for "head"}}

  <link integrity="" rel="stylesheet" href="{{rootURL}}assets/vendor.css">
```

```
  <link integrity="" rel="stylesheet" href="{{rootURL}}assets/assessment-app.css">
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
  <link
    href="https://fonts.googleapis.com/css2?family=Poppins:ital,wght@0,100;0,200;0,300;0,400;0,500;0,600;0,700;0,800;0,900;1,100;1,200;1,300;1,400;1,500;1,600;1,700;1,800;1,900&display=swap"
    rel="stylesheet">
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">

  {{content-for "head-footer"}}
</head>

<body>
  {{content-for "body"}}

  <script src="{{rootURL}}assets/vendor.js"></script>
  <script src="{{rootURL}}assets/assessment-app.js"></script>

  {{content-for "body-footer"}}
</body>

</html>
```

Templates/application.hbs

```
{{page-title "Assessment-App"}}

{{outlet}}
```

```
<span class="head-text">Digital Wallet &amp;amp; Subscriptions</span><br>
<span class="head-desc-text">Track your wallets and subscriptions</span>

<div class="container">
  <div class="left">
    <div class="my-wallet">
      <h2>My Wallet</h2>
      <p class="amount-title">Current Balance</p>
      <span class="amount"><i class="bi bi-currency-rupee"></i>{{this.amount}}</span>
      <div class="wallet-btn-container">
        <button class="wallet-btn withdraw" type="button">Withdraw</button>
        <button class="wallet-btn add-money" type="button" {{on "click"
this.toggleModal}}><i
            class="bi bi-wallet"></i> Add Money</button>
      </div>
    </div>

    <div class="transaction-history">
      <div class="transaction-header">
        <span class="heading">Transaction History</span>
        <button class="filter" {{on "click" this.toggleFilter}}><i class="bi bi-funnel"></i>
Filter</button>
        {{#if this.isFilterOpen}}
        <div class="filter-modal">
          <input type="checkbox" value="debit" name="filter-check" id="debit"><label
            for="debit">Debit</label><br>
          <input type="checkbox" value="refund" name="filter-check" id="refund"><label
            for="refund">Refund</label><br>
```

```
            <input type="checkbox" value="credit" name="filter-check" id="credit"><label
              for="credit">Credit</label>
            <div>
              <button {{on "click" this.clearFilter}} class="filter-modal-btn">Clear</button>
              <button {{on "click" this.applyFilter}} class="filter-modal-btn">Apply</button>
            </div>

          </div>
          {{/if}}
        </div>
        {{#each this.filteredHistory as |hist|}}
        <div class="transaction">
          <div>
            <span class="hist-name">{{hist.name}}</span> <br>
            <span style="font-size: 10px;">{{hist.subdate.date}} {{hist.subdate.month}}
{{hist.subdate.year}} {{hist.subdate.hour}}:{{hist.subdate.minute}}</span>
          </div>
          <span class="hist-desc">{{hist.transaction}}</span>
          <span class={{if (eq hist.transaction "Debit" ) "del" "plus" }}>{{if (eq hist.transaction
            "Debit") "-" "+"}}{{hist.subamount}}</span>
        </div>
        {{else}}
        <div class="empty-transaction">
          <img src="/notran.png" alt="img" style="width: 150px; height:150px;
margin:2rem;">
          <p style="font-size: 18px; font-weight:600; margin:0;">No transactions found</p>
        </div>
        {{/each}}
      </div>
```

```
    {{#if this.isModelOpen}}
    <div class="overlay"></div>
    <div class="add-modal">
      <i class="bi bi-x-lg close-icon" {{on "click" this.toggleModal}}></i>
      <label for="amount-value">Amount</label>
      <input type="number" placeholder="Enter Amount" id="amount-value">
      <button class="add-btn" type="button" {{on "click" this.addToWallet}}>Add to Wallet</button>
    </div>

    {{/if}}
  </div>

  <div class="subscription-tracker">
    <div class="subscription-tracker-header">
      <div class="subscription-tracker-head">
        <span class="heading">Subscription Tracker</span>
        <span class="description">Your subscriptions</span>
      </div>
      <button class="add-sub-btn" type="button" {{on "click" this.toggleSubModal}}>Add Subscription</button>
    </div>

    <div class="subscriptions">
      {{#each this.subscriptions as |subscription index|}}
      <div class="subscription-card">
        <i class="bi bi-pencil-square edit-icon" {{on "click" (fn this.editSubscription index)}}></i>
```

```
            <i class="bi bi-trash edit-icon del" {{on "click" (fn this.dltSubscription index)}}></i>
            <button class="edit-icon cancel" {{on "click" (fn this.cancelSubscription
index)}}>Cancel
              Subscription</button>
            <span class="name">{{subscription.name}}</span>
            <span class="sub-plan">{{subscription.subplan}}</span>
            <div>
              <span class="sub-amount"><i class="bi bi-currency-rupee"></i>
{{subscription.subamount}}</span><span
                style="color: #6c6c6c;">/</span>
              <span class="billing-cycle">{{subscription.billcycle}}</span>
            </div>
            <div>
              <span class="category">{{subscription.category}}</span>
              <span class="payment-method">{{subscription.paymethod}}</span>
              <span class="valid-date">Valid till {{subscription.validity.date}}
{{subscription.validity.month}}
{{subscription.validity.hour}}:{{subscription.validity.minute}}</span>
            </div>
          </div>

          {{else}}
          <div class="empty-subscription">
            <img src="/nosub.png" alt="img" style="width: 300px; height:300px;
margin:1rem">
            <p style="font-size: 18px; font-weight:600; margin:0;">No Subscriptions Found</p>
          </div>
          {{/each}}
```

```
    </div>


    {{#if this.isSubModelOpen}}
    <div class="overlay"></div>
    <div class="add-sub-modal">
      <i class="bi bi-x-lg close-icon" {{on "click" this.toggleSubModal}}></i>
      <label for="amount-value">Name</label>
      <input type="text" value={{this.name}} placeholder="Enter Name" id="name" {{on
"input" this.updateName}}>
      <PowerSelect @selected={{this.subplan}} @options={{this.plans}}
@onChange={{this.updatePlan}}
        @labelText="Subscription Plan" as |plan|>
        {{plan}}
      </PowerSelect>
      <PowerSelect @selected={{this.billcycle}} @options={{this.billings}}
@onChange={{this.updateBilling}}
        @labelText="Billing Cycle" as |billing|>
        {{billing}}
      </PowerSelect>
      <label for="sub-amount">Amount</label>
      <input type="number" value={{this.subamount}} placeholder="Enter Amount"
id="sub-amount" {{on "input"
        this.updateAmount}}>
      <PowerSelect @selected={{this.category}} @options={{this.categories}}
@onChange={{this.updateCategory}}
        @labelText="Category" as |category|>
        {{category}}
      </PowerSelect>
      <PowerSelect @selected={{this.paymethod}} @options={{this.paymentMethod}}
@onChange={{this.updatePayment}}
```

```
            @labelText="Payment Method" as |pay|>
            {{pay}}
          </PowerSelect>
          <button class="add-sub-btn add" type="button" {{on "click"
this.addSubscription}}>{{if this.isEditing
            "Update Subscription" "Add Subscription"}} </button>
        </div>
        {{/if}}

    </div>
</div>
<BasicDropdownWormhole />
```

Controllers/application.js

```
import Controller from '@ember/controller';
import { tracked } from '@glimmer/tracking';
import { action } from '@ember/object'
import { task, timeout } from "ember-concurrency"

export default class ApplicationController extends Controller {

    constructor() {
        super(...arguments)
        this.loadAmount();
        this.loadSubscription();
        this.loadHistory();
        this.checkSubscription.perform();
    }
```

```
    @tracked amount = 0;
    @tracked isModelOpen = false;
    @tracked isSubModelOpen = false;
    @tracked isEditing = false;
    @tracked editIndex = null;
    @tracked isFilterOpen = false;

    @tracked subscriptions = [];
    @tracked history = [];
    @tracked filteredHistory = [];

    @tracked name = '';
    @tracked subplan = '';
    @tracked billcycle = '';
    @tracked subamount = '';
    @tracked category = '';
    @tracked paymethod = '';

    months = ["Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov",
"Dec"];
    plans = ['Standard', 'Pro', 'Pro+'];
    billings = ['Weekly', 'Monthly', 'Yearly', 'Minute', 'Seconds'];
    categories = ['Entertainment', 'Music', 'Amazon Prime', 'Jio Star'];
    paymentMethod = ['UPI', 'Debit Card', 'Net Banking', 'Wallet'];

    @action toggleModal() {
        this.isModelOpen = !this.isModelOpen;
```

```
    }

    @action toggleSubModal() {
        this.isSubModelOpen = !this.isSubModelOpen;
        this.resetInputs();
    }

    @action toggleFilter() {
        this.isFilterOpen = !this.isFilterOpen;
    }

    @action addToWallet() {
        const value = document.getElementById('amount-value').value;
        if (value == '') {
            alert("Enter the amount");
            return;
        }
        let credit = {
            name: "Wallet",
            subamount: value,
            transaction: "Credit",
            subdate: {
                date: new Date().getDate(),
                day: new Date().getDay(),
                hour: new Date().getHours(),
                minute: new Date().getMinutes() < 10 ? "0" + new Date().getMinutes() : new
Date().getMinutes(),
                seconds: new Date().getSeconds(),
```

```
            month: this.months[new Date().getMonth()],
            year: new Date().getFullYear(),
        }
    }

    this.amount = parseInt(this.amount) + parseInt(value);
    this.history = this.history ? [...this.history, credit] : [credit];
    localStorage.setItem("history", JSON.stringify(this.history));
    localStorage.setItem("amount", this.amount);
    this.loadHistory();
    this.toggleModal();
}

@action updateName(event) {
    this.name = event.target.value;
}

@action updatePlan(plan) {
    this.subplan = plan;
}

@action updateBilling(bill) {
    this.billcycle = bill;
}

@action updateAmount(event) {
    this.subamount = event.target.value;
}
```

```
    @action updateCategory(cat) {
        this.category = cat;
    }

    @action updatePayment(pay) {
        this.paymethod = pay;
    }

    @action validitySetter(newDate) {
        return {
            dateObject: newDate,
            date: newDate.getDate(),
            day: newDate.getDay(),
            hour: newDate.getHours(),
            minute: newDate.getMinutes() < 10 ? "0" + newDate.getMinutes() :
newDate.getMinutes(),
            seconds: newDate.getSeconds(),
            month: this.months[newDate.getMonth()],
            year: newDate.getFullYear(),
        }
    }

    @action addSubscription() {
        let subscription = {
            name: this.name,
            subplan: this.subplan,
            billcycle: this.billcycle,
```

```
        subamount: this.subamount,
        category: this.category,
        paymethod: this.paymethod
      }

      if (!this.isEditing) {
        if (subscription.paymethod == "Wallet") {
          if (parseInt(this.subamount) <= parseInt(this.amount)) {
            this.amount = this.amount - this.subamount;
            localStorage.setItem("amount", this.amount);
            subscription.transaction = "Debit";
            subscription.subdate = {
              date: new Date().getDate(),
              day: new Date().getDay(),
              hour: new Date().getHours(),
              minute: new Date().getMinutes() < 10 ? "0" + new Date().getMinutes() : new
Date().getMinutes(),
              seconds: new Date().getSeconds(),
              month: this.months[new Date().getMonth()],
              year: new Date().getFullYear(),
            };
            if (subscription.billcycle == "Monthly") {
              var newDate = new Date(new Date().setDate(new Date().getDate() + 28));
              subscription.validity = this.validitySetter(newDate);

            } else if (subscription.billcycle == "Weekly") {
              var newDate = new Date(new Date().setDate(new Date().getDate() + 7));
              subscription.validity = this.validitySetter(newDate);
```

```
            } else if (subscription.billcycle == "Yearly") {
                var newDate = new Date(new Date().setDate(new Date().getDate() + 365));
                subscription.validity = this.validitySetter(newDate);

            } else if (subscription.billcycle == "Minute") {
                var newDate = new Date(new Date().setMinutes(new Date().getMinutes() + 1));
                subscription.validity = this.validitySetter(newDate);

            } else if (subscription.billcycle == "Seconds") {
                var newDate = new Date(new Date().setSeconds(new Date().getSeconds() + 5));
                console.log(newDate);
                subscription.validity = this.validitySetter(newDate);
            }

            this.subscriptions = this.subscriptions ? [...this.subscriptions, subscription] :
[subscription];
            localStorage.setItem("subscriptions", JSON.stringify(this.subscriptions));
            this.loadSubscription();
            this.toggleSubModal();
            this.resetInputs();
            this.history = this.history ? [...this.history, subscription] : [subscription];
            localStorage.setItem("history", JSON.stringify(this.history));
            this.loadHistory();
          } else {
            alert("Insufficient Wallet Balance");
          }
        } else {
```

```
            this.subscriptions = this.subscriptions ? [...this.subscriptions, subscription] :
[subscription];;
            localStorage.setItem("subscriptions", JSON.stringify(this.subscriptions));
            this.loadSubscription();
            this.toggleSubModal();
            this.resetInputs();
          }

      } else {
          this.subscriptions[this.editIndex] = subscription;
          this.isEditing = !this.isEditing;
          localStorage.setItem("subscriptions", JSON.stringify(this.subscriptions))
          this.toggleSubModal();
          this.loadSubscription();
          this.resetInputs();
      }
    }

    @action editSubscription(index) {
      this.editIndex = index;
      this.isEditing = !this.isEditing;
      this.toggleSubModal();
      this.name = this.subscriptions[index].name;
      this.subplan = this.subscriptions[index].subplan;
      this.billcycle = this.subscriptions[index].billcycle;
      this.subamount = this.subscriptions[index].subamount;
      this.category = this.subscriptions[index].category;
      this.paymethod = this.subscriptions[index].paymethod;
```

```
}

@action dltSubscription(index) {
    this.subscriptions.splice(index, 1);
    this.subscriptions = [...this.subscriptions];
    localStorage.setItem("subscriptions", JSON.stringify(this.subscriptions));
    this.loadSubscription();

}

@action cancelSubscription(index) {
    let cancelled = {
        name: this.subscriptions[index].name,
        subplan: this.subscriptions[index].subplan,
        billcycle: this.subscriptions[index].billcycle,
        subamount: this.subscriptions[index].subamount,
        category: this.subscriptions[index].category,
        paymethod: this.subscriptions[index].paymethod,
        transaction: "Refund"
    }

    if (this.subscriptions[index].paymethod == "Wallet") {
        this.amount = parseInt(this.amount) + parseInt(this.subscriptions[index].subamount);
        localStorage.setItem("amount", this.amount);
        cancelled.subdate = {
            date: new Date().getDate(),
            day: new Date().getDay(),
            hour: new Date().getHours(),
```

```
            minute: new Date().getMinutes() < 10 ? "0" + new Date().getMinutes() : new
Date().getMinutes(),
            seconds: new Date().getSeconds(),
            month: this.months[new Date().getMonth()],
            year: new Date().getFullYear(),
          };
          this.history = this.history ? [...this.history, cancelled] : [cancelled];
          localStorage.setItem("history", JSON.stringify(this.history));
          this.loadHistory();
          alert("Amount refunded to wallet");
        }
        this.subscriptions.splice(index, 1);
        this.subscriptions = [...this.subscriptions];
        localStorage.setItem("subscriptions", JSON.stringify(this.subscriptions));
        this.loadSubscription();
    }

    @action applyFilter() {
        let checkboxes = Array.from(document.querySelectorAll('input[name="filter-
check"]:checked'));
        let checked = checkboxes.map(cb => cb.value)
        if (checked == '') {
            alert("Select a filter");
            return;
        }
        this.filteredHistory = this.history.filter(h =>
checked.includes(h.transaction.toLowerCase()));
        this.toggleFilter();
    }
```

```
@action clearFilter() {
    let cbs = document.querySelectorAll('input[name="filter-check"]:checked')
    cbs.forEach(cb => cb.checked = false);
    this.filteredHistory = this.history;
    this.toggleFilter();
}

@action resetInputs() {
    this.name = '';
    this.subplan = '';
    this.billcycle = '';
    this.subamount = '';
    this.category = '';
    this.paymethod = '';
}

@action loadAmount() {
    const resAmount = localStorage.getItem("amount") || 0;
    this.amount = resAmount;
}

@action loadSubscription() {
    const resSubscription = JSON.parse(localStorage.getItem("subscriptions"));
    this.subscriptions = resSubscription;
}

@action loadHistory() {
```

```
        const resHistory = JSON.parse(localStorage.getItem("history"));
        this.history = resHistory;
        this.filteredHistory = this.history
    }

    @action setToLocalStorage(subs) {
        this.amount = this.amount - subs.subamount;
        localStorage.setItem("amount", this.amount);
        localStorage.setItem("subscriptions", JSON.stringify(this.subscriptions));
        this.loadSubscription();
        this.history = this.history ? [...this.history, subs] : [subs];
        localStorage.setItem("history", JSON.stringify(this.history));
        this.loadHistory();
    }

    @task *checkSubscription() {
        while (true) {
            console.log("Running")
            this.subscriptions.forEach(subs => {
                if (subs.billcycle == "Monthly" && new Date(subs.validity.dateObject) < new Date())
{
                    alert(`${subs.name} Subscription Updated`)
                    var newDate = new Date(new Date().setDate(new Date().getDate() + 28));
                    subs.subdate = {
                        date: new Date().getDate(),
                        day: new Date().getDay(),
                        hour: new Date().getHours(),
                        minute: new Date().getMinutes() < 10 ? "0" + new Date().getMinutes() : new
Date().getMinutes(),
```

```
                seconds: new Date().getSeconds(),
                month: this.months[new Date().getMonth()],
                year: new Date().getFullYear(),
            };
            subs.validity = this.validitySetter(newDate);
            this.setToLocalStorage(subs);
        }
        else if (subs.billcycle == "Weekly" && new Date(subs.validity.dateObject) < new
Date()) {
            alert(`${subs.name} Subscription Updated`)
            var newDate = new Date(new Date().setDate(new Date().getDate() + 7));
            subs.subdate = {
                date: new Date().getDate(),
                day: new Date().getDay(),
                hour: new Date().getHours(),
                minute: new Date().getMinutes() < 10 ? "0" + new Date().getMinutes() : new
Date().getMinutes(),
                seconds: new Date().getSeconds(),
                month: this.months[new Date().getMonth()],
                year: new Date().getFullYear(),
            };
            subs.validity = this.validitySetter(newDate);
            this.setToLocalStorage(subs);

        }
        else if (subs.billcycle == "Yearly" && new Date(subs.validity.dateObject) < new
Date()) {
            alert(`${subs.name} Subscription Updated`)
            var newDate = new Date(new Date().setDate(new Date().getDate() + 365));
```

```
            subs.subdate = {
              date: new Date().getDate(),
              day: new Date().getDay(),
              hour: new Date().getHours(),
              minute: new Date().getMinutes() < 10 ? "0" + new Date().getMinutes() : new
Date().getMinutes(),
              seconds: new Date().getSeconds(),
              month: this.months[new Date().getMonth()],
              year: new Date().getFullYear(),
            };
            subs.validity = this.validitySetter(newDate);
            this.setToLocalStorage(subs);
          }
          else if (subs.billcycle == "Minute" && new Date(subs.validity.dateObject) < new
Date()) {
            alert(`${subs.name} Subscription Updated`)
            var newDate = new Date(new Date().setMinutes(new Date().getMinutes() + 1));
            subs.subdate = {
              date: new Date().getDate(),
              day: new Date().getDay(),
              hour: new Date().getHours(),
              minute: new Date().getMinutes() < 10 ? "0" + new Date().getMinutes() : new
Date().getMinutes(),
              seconds: new Date().getSeconds(),
              month: this.months[new Date().getMonth()],
              year: new Date().getFullYear(),
            };
            subs.validity = this.validitySetter(newDate);
            this.setToLocalStorage(subs);
```

```
            }
            else if (subs.billcycle == "Seconds" && new Date(subs.validity.dateObject) < new
Date()) {
                var newDate = new Date(new Date().setSeconds(new Date().getSeconds() + 5));
                subs.subdate = {
                    date: new Date().getDate(),
                    day: new Date().getDay(),
                    hour: new Date().getHours(),
                    minute: new Date().getMinutes() < 10 ? "0" + new Date().getMinutes() : new
Date().getMinutes(),
                    seconds: new Date().getSeconds(),
                    month: this.months[new Date().getMonth()],
                    year: new Date().getFullYear(),
                };
                subs.validity = this.validitySetter(newDate);
                this.setToLocalStorage(subs);
                alert(`${subs.name} Subscription Updated`)
            }
        })

        yield timeout(10000);
    }
  }
}
```

Styles/app.css

```
/* Ember supports plain CSS out of the box. More info:
https://cli.emberjs.com/release/advanced-use/stylesheets/ */

body {
  font-family: Poppins;
  margin: 1.5rem 2rem;
  color: #2a2a2a;
}

.overlay {
  width: 100%;
  height: 100%;
  background-color: #2a2a2a;
  opacity: 0.7;
  position: fixed;
  top: 0;
  left: 0;
  z-index: 1;
}

.head-text {
  font-size: 26px;
  font-weight: 500;
}

.head-desc-text, .description {
  font-size: 16px;
  color: #676767;
```

```
}

.container {
    display: flex;
    flex-wrap: wrap;
    gap: 2rem;
    margin: 2rem 0;

}
.left {
    display: flex;
    flex-direction: column;
    width: 30%;
}

.my-wallet {
    background:linear-gradient(90deg,rgb(42 123 155 / 100%) 0%, rgb(87 199 133 / 100%) 50%, rgb(237 221 83 / 100%) 100%);
    color: #fff;
    border-radius: 10px;
    padding: 1.5rem;
    height: 240px;
}

.amount-title {
    color: #2b3e47;
}
```

```
.amount {
    font-size: 48px;
}

.wallet-btn-container {
    display: flex;
    gap: 1rem;
    margin: 0.5rem 0;
}

.wallet-btn {
    border: none;
    padding: 0.8rem 2rem;
    border-radius: 10px;
    background-color: #fcfcfc;
    color: #353535;
    cursor: pointer;
    font-weight: 700;
}

.wallet-btn i {
    font-weight: 800;
}

.add-modal {
    background-color: #fff;
    color: black;
    border-radius: 10px;
```

```
    padding: 2rem 4rem;
    z-index: 5;
    position: fixed;
    top: 30%;
    left: 40%;
    display: flex;
    flex-direction: column;
    gap: 1rem;
}

input {
    padding: 0.5rem 1rem;
    border-radius: 5px;
    border: 1px solid #ccc;
    width: auto;

}

.close-icon {
    position: absolute;
    top: 1rem;
    right: 1rem;
    cursor: pointer;
}

.filter {
    border: 1px solid #ccc;
    background: transparent;
```

```
    border-radius: 10px;
    padding: 0.5rem 1rem;
    cursor: pointer;
    font-weight: 600;
    color: #2a2a2a;
}

.filter-modal {
    position: absolute;
    background-color: #fff;
    border: 1px solid #ccc;
    padding: 1rem;
    border-radius: 10px;
    top: 4.5rem;
    right: 1rem;
    box-shadow: 2px 4px 6px rgba(0,0,0,0.1);
}

.filter-modal label {
    font-size: 14px;
}

.filter-modal-btn {
    border: 1px solid #ccc;
    border-radius: 10px;
    background: transparent;
    padding: 0.3rem 1rem;
    cursor: pointer;
```

```
    margin-top: 1rem;
}

input[type="checkbox"] {
    margin-right: 0.5rem;
}

.transaction-history {
    margin-top: 1rem;
    padding: 2rem;
    background-color: #fff;
    box-shadow: 1px 2px 4px rgb(0 0 0 / 20%);
    border: 1px solid #ddd;
    position: relative;
}

.transaction {
    border-bottom: 1px solid #ccc;
    margin: 1.5rem auto;
    display: flex;
    justify-content: space-between;
    align-items: center;
    gap: 3rem;
}

.transaction-header {
    display: flex;
    justify-content: space-between;
```

```
}

.hist-name {
    font-size: 16px;
}

.hist-desc {
    font-size: 14px;
    color: #575757;
}

.subscription-tracker {
    background-color: #fff;
    box-shadow: 1px 2px 4px rgb(0 0 0 / 20%);
    border: 1px solid #ddd;
    padding: 2rem;
    display: flex;
    flex-direction: column;
    width: 60%;
}

.add-btn, .add-sub-btn {
    border: none;
    padding: 0.6rem 1rem;
    background-color: #2a2a2a;
    color: white;
    border-radius: 5px;
    cursor: pointer;
```

```
}

.add {
    margin: 1rem 0;
}

.subscription-tracker-header {
    display: flex;
    align-items: start;
    justify-content: space-between;
    gap: 2rem;
    margin-bottom: 1.5rem;
}

.subscription-tracker-head {
    display: flex;
    flex-direction: column;
}

.heading {
    font-size: 22px;
    font-weight: 600;
}

.subscriptions {
    display: flex;
    gap: 1.5rem;
    flex-wrap: wrap;
```

```
}

.subscription-card {
    padding: 1.5rem;
    width: 42%;
    border-radius: 20px;
    display: flex;
    flex-direction: column;
    justify-content: center;
    align-items: start;
    box-shadow: 1px 2px 4px rgba(0,0,0,0.3);
    position: relative;
    border: 1px solid #ddd;
}

.subscription-card div {
    margin: 1.5rem 0;
}

.edit-icon {
  position: absolute;
  top: 1rem;
  right: 3rem;
  cursor: pointer;
}

.del {
    right: 1rem;
```

```
    color: red;
}

.plus {
    color: green;
}

.cancel {
    right: 5rem;
    border: 1px solid #e64646;
    border-radius: 10px;
    padding: 0.3rem 0.8rem;
    background: transparent;
    color: #e64646;
    font-size: 12px;
}


.add-sub-modal {
    background-color: #fff;
    color: black;
    border-radius: 10px;
    padding: 2rem;
    z-index: 5;
    position: fixed;
    top: 5%;
    display: flex;
    flex-direction: column;
```

```
    gap: 1rem;
    width: 400px;
}

.name {
    font-size: 22px;
    font-weight: 500;
}

.sub-plan {
    font-size: 14px;
    color: #666666;
}

.sub-amount {
    font-size: 30px;
}

.billing-cycle {
    font-size: 14px;
    color: #6c6c6c;
}

.category, .payment-method, .valid-date {
    border-radius: 20px;
    padding: 0.3rem 1rem;
    font-size: 12px;
}
```

```
.category {
    background-color: #ccc;
    margin-right: 0.5rem;
}

.payment-method {
    background-color: #4cad6e;
    color: white;
}

.valid-date {
    border: 1px solid #ccc;
}

.empty-subscription, .empty-transaction {
    display: flex;
    justify-content: center;
    align-items: center;
    flex-direction: column;
    width: 100%;
    /* height: 50vh; */
}
```