`rm db.sqlite3` Fresh dev DB
`source venv/bin/activate`

`python manage.py makemigrations contracts` - keep the python model definitions and the underlying tables in sync / Django records that change in a new migration

`python manage.py migrate` so your database picks up the new schema.

`python manage.py loaddata base_contract` tells django to lead the fixture

PRINT LIST MARKUP

```
      {% for service in service_types %}
      <input type="hidden" name="service_types" value="{{ service }}">
      {% endfor %}
      {% if service_types %}
      <div class="th-pill-row">
        {% for service in service_types %}
        <span class="th-pill">{{ service }}</span>
        {% endfor %}
      </div>
      {% endif %}
```

PRINT LIST VIEWS

```
if 'add' in request.GET and form.is_valid():
          new_service = form.cleaned_data["service_type"]
          if new_service not in service_types:
              service_types.append(new_service)
```

```
value="{{ search_term|default_if_none:'' }}"
```

Security:
How do you avoid exposing internal contract IDs? Would you limit rate lookup access or add throttling/audit logging?

Performance:
For bulk lookups, how would you optimize if there are hundreds of service types? Would you paginate, schedule background jobs, or cache results?

““How would you extend get_service_rate_on_date to support querying across multiple contracts”

The problem with this approach is:

It reissues a full query for every (contract_id, service_type) pair which will hit the DB separately and waste work reloading the same contract, base rates, and amendment rates repeatedly.

Also the whole batch could be stopped on the first error.

The naïve loop should take a grouped approach that validates inputs, fetches each contract (with base rates + amendment rates) only once, and reuses that data to answer every request for the shared query_date.