Test Project

Session 6 - CivicCharge

IT Software Solutions for Business

Submitted by:
Faye FEI - Neusoft Education and Technology Group, Independent Test Project Designer
Benjamin Frost, Independent Test Project Validator

Contents

Introduction to Test Project **3**
Instructions to the Competitor **3**
Other Requirements and Notes **3**
Description of project and tasks **3**
2.1 Nearby Services 3
2.2 Review Management 7
2.3 User Profile 8
2.3.1 Personal Review Management *9*
2.4 Map Navigation 11
2.4.1 Map Display with Zoom Functionality *11*
2.4.2 Set Start and Destination coordinates *11*

© WorldSkills International

Introduction

CivicCharge is a non-profit that runs a city's **kerbside EV-charging network** on behalf of residents who have no off-street parking. Its back office bills charging sessions against a **time-of-use tariff**: electricity costs more at peak times, so the price of a session depends on **when** the energy was delivered, plus an **idle fee** for cars that stay plugged in after charging finishes.

You have joined the team that maintains `CivicCharge.Billing` — the headless, deterministic library at the heart of the back office. It works for the common case, but support has logged a backlog of **defect tickets**, and product has **three feature requests** on top. Your job this session is to **work the backlog**: fix the reported defects and deliver the requested features, **without breaking behaviour that already works**.

Description of project and tasks

Resolve the **defect tickets** and **feature requests** below by changing code in **`CivicCharge.Billing.Services`** only. You may edit, refactor, or rewrite anything inside `Services`; you may **not** modify the `Contracts` package or the tests. The library stays pure, deterministic, and headless — no UI, database, clock, randomness, or I/O. Every rule needed to decide what **correct** looks like is given below, so **no energy-industry background is required**.

Conventions

Money is integer **cents**; energy is integer **watt-hours (Wh)**; a time is an integer **minute of the day** in `[0, 1440)`; a tariff rate is integer **cents per kWh**. Sessions fall within a single day. The data types (immutable `record`s in `Contracts`) and the tariff-window notation used below are defined in the appendices at the end.

1. **Determinism.** No randomness, clock, culture-dependent parsing, or I/O. Same input ⇒ same output.
2. **Never throw.** No method may throw for any input. On invalid/degenerate input, return the **documented sentinel**. Throwing fails the relevant tests.
3. **Charging vs idle.** Energy is delivered, at constant power, over `[StartMinute, ChargeCompleteMinute)`. The bay is occupied until `EndMinute`; idle time is `EndMinute – ChargeCompleteMinute`.
4. **Rounding.** Round half away from zero (`MidpointRounding.AwayFromZero`). Per-window energy is conserved (see the energy split below); cents are whole numbers.
5. **Collections.** A `null` collection argument is treated as empty; never return `null` for a collection.
6. **Identifiers.** All `Id` comparisons are **case-sensitive ordinal** string matches; report rows sort by `ConnectorId` **ordinal ascending**.
7. **Marking tolerance.** Ids, issue codes, counts, and cents are compared **exactly**.

The system you inherited (intended behaviour)

These are the rules the library is **supposed** to follow — your reference for what counts as a defect. Components depend on each other in order; **Component E** orchestrates A–D and receives them by constructor injection.

© WorldSkills International

Component A — ITariffValidator

```
ValidationResult Validate(Tariff tariff);
```

Return an ordered list of **issue codes**; the tariff is valid only when the list is empty. Evaluate in order, appending each code the first time its rule is violated (no duplicates):

ORDER	ISSUE CODE	CONDITION
1	empty-tariff	tariff is null, or Windows is null/empty
2	window-out-of-range	a window has StartMinute < 0, EndMinute > 1440, or StartMinute >= EndMinute
3	windows-overlap	two windows overlap — i.e. they share any minute under half-open [Start, End) (windows that merely *touch* are fine)
4	windows-not-tiling	in start order the windows do not cover [0, 1440) exactly — a gap between one window's end and the next window's start, or coverage not starting at 0 / not ending at 1440
5	negative-rate	a window has RatePerKwhCents < 0
6	invalid-fees	IdleFeePerMinuteCents < 0, GraceMinutes < 0, or SessionCapCents present and < 0

- **Sentinel:** never throw; always return a ValidationResult. A null tariff ⇒ { IsValid: false, Issues: ["empty-tariff"] }.

Component B — IEnergyAllocator

```
IReadOnlyList<WindowCharge> Allocate(ChargingSession session, Tariff tariff);
```

Split the energy across the tariff windows the **charging** interval overlaps, and price each slice. The symbols used below:

- M — the charging length in minutes (ChargeCompleteMinute – StartMinute).
- a_w, b_w — window w's StartMinute and EndMinute (it covers [a_w, b_w)).
- m_w — the minutes of the charging interval that fall inside window w.
- $energy_w$ — the Wh apportioned to window w.
- $cost_w$ — window w's cost, in cents.

Charging runs over the half-open interval from StartMinute to ChargeCompleteMinute; call its length M minutes. For a window w covering [a_w, b_w), the minutes inside the charging interval are

$$m_w = max\big(0,\ min(ChargeComplete,\ b_w) - max(Start,\ a_w)\big)$$

(half-open, so a boundary minute belongs to **exactly one** window). The raw energy share of a window is

$$rawEnergy_w = EnergyWh \cdot m_w / M$$

© WorldSkills International

Floor each share, then **conserve the total** — hand the leftover Wh one at a time to the windows with the largest fractional parts, ties broken by **lowest window start** — so the slices sum to `EnergyWh`. Each slice's cost converts the window's `RatePerKwhCents` and its Wh to whole cents (half away from zero):

$$cost_w = round(RatePerKwhCents_w \cdot energy_w / 1000)$$

Return one `WindowCharge` per window with $m_w > 0$, ordered by window start.

- **Sentinel:** `EnergyWh <= 0`, `M <= 0`, or a null/empty tariff ⇒ empty list. Never throw.

Example. Charging `08:00–08:40` (`Start = 480`, `ChargeComplete = 520`, `M = 40`), `EnergyWh = 8000`, against a peak window `[480, 510)` at 50 ¢/kWh and an off-peak window `[510, 1440)` at 20 ¢/kWh: `30` min peak ⇒ `6000` Wh ⇒ `300` ¢, `10` min off-peak ⇒ `2000` Wh ⇒ `40` ¢.

Component C — `ISessionPricer`

```
SessionCost Price(ChargingSession session, Tariff tariff);
```

`EnergyCostCents` is the sum of the slice costs. **Member discount** (FR-3): if `session.IsMember`, reduce the energy cost — and only the energy cost — by `MemberDiscountBasisPoints` (`10000` = 100%): `discount = round(EnergyCostCents × MemberDiscountBasisPoints / 10000)`, applied **before** the cap. Idle: `idle = max(0, EndMinute – ChargeCompleteMinute)`, `billable idle = max(0, idle – GraceMinutes)`, `IdleFeeCents = billableIdle × IdleFeePerMinuteCents`. `TotalCents = (EnergyCostCents – discount) + IdleFeeCents`, then **capped to `SessionCapCents`** when present (`TotalCents = min(TotalCents, cap)`). `Breakdown` is the per-window result.

- **Sentinel:** a degenerate/invalid session or tariff ⇒ a zeroed `SessionCost` for that session id. Never throw.

Component D — `ISiteDailyReporter`

```
IReadOnlyList<ConnectorDaySummary> Summarise(IReadOnlyList<ChargingSession> sessions, Tariff tariff);
```

Group sessions by `ConnectorId`; per connector report the session count, total `EnergyWh`, total revenue (`Σ TotalCents`, priced per session), and occupied minutes (`Σ (EndMinute – StartMinute)`). Order rows by `ConnectorId` ordinal ascending.

- **Sentinel:** null/empty `sessions` ⇒ empty list. Never throw.

Component E — `IBillingRun`

```
IReadOnlyList<SessionCost> Run(IReadOnlyList<ChargingSession> sessions, Tariff tariff);
```

If `Validate(tariff)` is **not** valid, return an **empty** list — never price against a broken tariff. Otherwise price every session, **in input order**, reusing the components above (injected; do not re-implement them).

- **Sentinel:** invalid tariff or null/empty sessions ⇒ empty list. Never throw.

© WorldSkills International

Reported defects (support backlog)

Below are the tickets as support filed them — reproduction steps, what they saw, and what they expected. The **root cause is for you to find**: reproduce each issue, work out from its behaviour which part of `Services` is responsible, and fix it against the rules above.

T-01 · Touching windows rejected as overlapping — Priority: Medium

- **Steps.** Validate a tariff with two windows `DAY [0, 720)` @20 ¢/kWh and `EVE [720, 1440)` @40 ¢/kWh, fees valid.
- **Observed.** `IsValid` is `false`; `Issues` contains `windows-overlap`.
- **Expected.** `IsValid` is `true`, `Issues` empty — windows that only *touch* at minute 720 share no minute under `[Start, End)`.

T-02 · Whole-day tariff flagged as not tiling — Priority: High

- **Steps.** Validate that same two-window tariff, which covers `[0, 1440)` exactly.
- **Observed.** `Issues` contains `windows-not-tiling`.
- **Expected.** No `windows-not-tiling` issue — coverage from 0 to 1440 with no gap is complete.

T-03 · Negative price accepted — Priority: High

- **Steps.** Validate a tariff containing a window with `RatePerKwhCents = -5` (everything else valid).
- **Observed.** `IsValid` is `true`; the negative rate passes silently.
- **Expected.** `Issues` contains `negative-rate`; `IsValid` is `false`.

T-04 · Boundary minute billed in both windows — Priority: High

- **Steps.** Price a session charging `11:50–12:10` (`Start 710`, `ChargeComplete 730`, `End 730`, `EnergyWh 4000`) on tariff `DAY [0,720)` @20 / `EVE [720,1440)` @40.
- **Observed.** Breakdown energies are `DAY 2200 Wh` and `EVE 2200 Wh` — summing to `4400`, i.e. `400` Wh more than delivered; minute 720 is counted in both windows.
- **Expected.** `DAY 2000 Wh` (10 min) and `EVE 2000 Wh` (10 min), summing to `4000`; costs `40` ¢ + `80` ¢.

T-05 · Energy lost on short multi-window sessions — Priority: Medium

- **Steps.** Price a small session `11:57–12:03` (`Start 717`, `ChargeComplete 723`, `EnergyWh 5`) crossing the 720 boundary (3 min `DAY`, 3 min `EVE`).
- **Observed.** Both slices show `0` Wh; total billed energy is `0`.
- **Expected.** Energy is apportioned without truncation, slices summing to `EnergyWh = 5` (e.g. `DAY 3`, `EVE 2`).

T-06 · One extra free idle minute — Priority: Medium

- **Steps.** Price a session that finished charging on arrival (`Start = ChargeComplete = 660`) and sat plugged in until `End = 680` — 20 idle minutes — with `GraceMinutes 10`, `IdleFeePerMinuteCents 5`.
- **Observed.** Idle fee is `45` ¢ — only 9 minutes are billed.
- **Expected.** Idle fee is `50` ¢ — billable idle is `max(0, 20 – 10) = 10` minutes.

T-07 · Idle billed from plug-in, not charge-complete — Priority: High

- **Steps.** Price a session `Start 600`, `ChargeComplete 660`, `End 668`, `GraceMinutes 10`, `5` ¢/min.
- **Observed.** Idle fee is `285` ¢ — it bills the whole time plugged in (`End – Start = 68` min).
- **Expected.** Idle fee is `0` ¢ — real idle is `End – ChargeComplete = 8` min, inside the 10-minute grace.

© WorldSkills International

T-08 · Session cap ignores the idle fee — Priority: High

- **Steps.** Price a session whose energy cost is `120 ¢` with an idle fee on top and `SessionCapCents 200`, so the uncapped bill exceeds 200.
- **Observed.** `TotalCents` comes back **above** 200 — the cap was applied to the energy cost alone, so the idle fee pushed the bill back over the limit.
- **Expected.** `TotalCents` is `200` — the cap bounds the **total**: `min(energyAfterDiscount + idleFee, 200)`.

T-09 · Daily report rows are not sorted — Priority: Medium

- **Steps.** Summarise sessions whose connectors first appear in the order `"Bay-2"`, `"Bay-10"`, `"Bay-1"`.
- **Observed.** Rows come back in that arbitrary input order (`"Bay-2"`, `"Bay-10"`, `"Bay-1"`).
- **Expected.** Rows are sorted by `ConnectorId` **ordinal ascending**: `"Bay-1"`, `"Bay-10"`, `"Bay-2"`.

T-10 · Occupied minutes ignore idle time — Priority: Medium

- **Steps.** Summarise a connector with one session `Start 600`, `ChargeComplete 660`, `End 700`.
- **Observed.** `OccupiedMinutes` is `60` — only the charging time is counted.
- **Expected.** `OccupiedMinutes` is `100` — the bay is occupied from `Start` to `End` (`End – Start`), including idle.

Feature requests

Written as user stories with acceptance criteria; implement them in `Services` alongside the defect fixes.

FR-1 — Multi-tier (N-window) tariffs · Priority: High

As a tariff manager, **I want** session energy priced correctly across **any number** of time-of-use windows — not just two — **so that** we can launch shoulder rates and dynamic multi-window tariffs without mispricing sessions.

Acceptance criteria.

1. **Given** a three-window tariff `shoulder [720,960) @25` / `peak [960,1200) @50` / `eve [1200,1440) @30` and a session charging `15:30–18:30 (930 → 1110)`, **when** priced, **then** the breakdown has one slice per overlapped window in start order — `shoulder` 30 min, `peak` 150 min — with slice energies summing to `EnergyWh`.
2. **Given** a session whose charging interval lies entirely within one window, **then** the breakdown has exactly one slice.
3. **Given** a session that finishes **exactly** on a window boundary, **then** no energy is attributed to the next window.
4. **Given** a session overlapping **four or more** windows, **then** every overlapped window appears, correctly apportioned, with energies summing to `EnergyWh`.

FR-2 — Free-charging promotions · Priority: Medium

As a campaign manager, **I want** windows priced at `0` ¢/kWh to bill as free while still showing on the receipt, **so that** we can run "free off-peak charging" promotions and drivers can see the energy they received at no cost.

Acceptance criteria.

1. **Given** a tariff with a `0` ¢/kWh window and a session charging during it, **when** priced, **then** the breakdown includes that window's slice with its apportioned `EnergyWh` and a `CostCents` of `0`.
2. **Given** a session charging **entirely** within a free window, **then** `EnergyCostCents` is `0` but the breakdown is **not** empty.

FR-3 — Member energy discount · Priority: Medium

As a CivicCharge member, **I want** a percentage discount on the **energy** part of my bill, **so that** membership makes charging cheaper while idle penalties still apply in full.

© WorldSkills International

Acceptance criteria.

1. **Given** a member session (`IsMember = true`) and `MemberDiscountBasisPoints = 1500`, **when** priced, **then** `EnergyCostCents` is reduced by 15% (rounded half away from zero) and `IdleFeeCents` is unchanged.
2. **Given** the discount **and** a `SessionCapCents`, **then** the discount is applied **before** the cap.
3. **Given** a non-member session (`IsMember = false`), **then** no discount is applied.

Note. `IsMember` and `MemberDiscountBasisPoints` already exist in `Contracts` but are not yet applied.

Instructions to the Competitor

This section contains the instructions given to the Competitor for this session. The instructions can be increased and improved at the Competition during the preparation time.

Other

Use this section for information that is important but is not covered in the previous sections.

Appendix — Tariff-window notation

A tariff window is written `Id [startMinute, endMinute)` followed by an `@` and a rate — for example `DAY [0, 720) @20`. It reads as:

- **Label** — `DAY`, `EVE`, `peak`, `shoulder` and the like are only names; they carry no meaning beyond identifying the window.
- **[a, b)** — a half-open range of minute-of-day, so `[0, 720)` covers 00:00–11:59 and `[720, 1440)` covers 12:00–23:59.
- **Rate** — the number after the `@` is the price for that window, in cents per kWh.

So `DAY [0, 720) @20` is the `DAY` window, 00:00–11:59, priced at 20 cents per kWh; `EVE [720, 1440) @40` is 12:00–23:59 at 40 cents per kWh.

© WorldSkills International

Appendix - Domain model

These are the immutable `record` types from `CivicCharge.Billing.Contracts` (field names are illustrative of the supplied types).

TariffWindow — a time-of-use pricing window, half-open over [StartMinute, EndMinute).

FIELD	TYPE	DESCRIPTION
Id	string	Window label, e.g. DAY or EVE.
StartMinute	int	Inclusive start minute of the day.
EndMinute	int	Exclusive end minute of the day.
RatePerKwhCents	int	Energy price for the window, in cents per kWh.

Tariff — the full pricing scheme for the day.

FIELD	TYPE	DESCRIPTION
Id	string	Tariff identifier.
Windows	IReadOnlyList<TariffWindow>	The pricing windows; a valid tariff tiles [0, 1440).
IdleFeePerMinuteCents	int	Idle fee per minute beyond grace, in cents.
GraceMinutes	int	Free idle minutes after charging completes.
SessionCapCents	int?	Optional cap on the session total, in cents (null = uncapped).
MemberDiscountBasisPoints	int	Member discount on energy cost, in basis points (10000 = 100%).

ChargingSession — one charging session at a connector.

FIELD	TYPE	DESCRIPTION
Id	string	Session identifier.
ConnectorId	string	The connector used.
StartMinute	int	Minute the session began occupying the bay.
ChargeCompleteMinute	int	Minute charging finished.
EndMinute	int	Minute the bay was vacated.
EnergyWh	int	Total energy delivered, in watt-hours.
IsMember	bool	True when the driver is a member.

© WorldSkills International

WindowCharge — the energy and cost attributed to one window.

FIELD	TYPE	DESCRIPTION
WindowId	string	The tariff window.
EnergyWh	int	Energy apportioned to this window.
CostCents	int	Cost for this window, in cents.

SessionCost — the priced result of one session.

FIELD	TYPE	DESCRIPTION
SessionId	string	The session.
EnergyCostCents	int	Sum of the window costs, before any discount.
IdleFeeCents	int	Idle fee charged.
TotalCents	int	Final total after discount, capped to SessionCapCents.
Breakdown	IReadOnlyList<WindowCharge>	The per-window charges.

ConnectorDaySummary — a per-connector daily rollup.

FIELD	TYPE	DESCRIPTION
ConnectorId	string	The connector.
Sessions	int	Number of sessions.
EnergyWh	int	Total energy delivered, in watt-hours.
RevenueCents	int	Total revenue, in cents.
OccupiedMinutes	int	Total bay-occupied minutes.

ValidationResult — the outcome of validating a tariff.

FIELD	TYPE	DESCRIPTION
IsValid	bool	True only when Issues is empty.
Issues	IReadOnlyList<string>	The issue codes, in the order they were raised.

© WorldSkills International