Naija Prime School

Sprint 4 — Attendance

Daily registers · Per-subject sessions · Submission lifecycle · Class summaries

Long-form implementation walk-through

Author: Benjamin Fadina

Branch: sprint/4-attendance

Built on: Sprint 1 identity + Sprint 2 academic domain + Sprint 3 students & parents

Stack: .NET 10, Blazor Web App (Auto), EF Core 10, SQL Server, Radzen Blazor

Editor: Visual Studio Code with the C# Dev Kit

Repository: https://github.com/benjaminsqlserver/NaijaPrimeSchool

Contents

1. Sprint 4 in context

Sprint 4 plugs attendance into the structure that sprints 1–3 laid down. It is the first sprint in which the application starts carrying actual day-to-day classroom data — every previous sprint set up the scaffolding, but nothing changed from one school day to the next. After sprint 4 ships, every classroom day generates a register; every lesson on the timetable can generate a per-subject session; and the school can pull a per-pupil attendance percentage for any term or session.

There are two parallel attendance models, deliberately built side by side. The daily register is what most Nigerian primary schools actually run on: at homeroom each morning, the class teacher marks every pupil Present, Absent, Late, Excused, Sick or Suspended in a single sheet. The subject register is finer-grained: it sits on top of the timetable from sprint 2 and lets a subject teacher take attendance at any specific (class × subject × period × day) slot. Both models share the same AttendanceStatus lookup, the same submit/reopen lifecycle, and the same auditing and soft-delete primitives the previous sprints established.

This document is a long-form implementation guide. It is written so that an engineer who has read the sprint 3 guide and has the codebase checked out can recreate every change in this sprint without referring to the diff. The document is organised in roughly the order I built the code in: design decisions first, Domain entities next, Application contracts after that, Infrastructure (DbContext, services, seeder, migration) in the middle, and finally the Razor UI and navigation. There is a smoke-test chapter near the end that walks through the happy-path you can use to confirm a fresh checkout works.

1.1 Where this sits relative to sprint 3

Sprint 3 delivered Student, Parent, StudentParent linkage, and Enrolment. Sprint 4 leans on every one of those:

- BaseEntity — every new attendance entity inherits from it and picks up Guid Id, IAuditable, and ISoftDelete with no boilerplate.
- ApplicationDbContext.SaveChanges — the override stamps CreatedOn/By and ModifiedOn/By and rewrites Delete to IsDeleted = true. Every attendance write therefore inherits auditing and soft delete with zero changes to the override.
- Global query filters — every new entity declares HasQueryFilter(x => !x.IsDeleted), so deleted rows vanish from ordinary queries without service code having to remember to filter.
- OperationResult / OperationResult<T> — every attendance service uses this for predictable success/failure responses.
- Enrolment — DailyAttendanceService.OpenAsync and SubjectAttendanceService.OpenAsync read open enrolments (WithdrawnOn IS NULL) to pre-populate the register with every pupil who is currently in the class on the date being marked. This is exactly the load-bearing definition the sprint 3 guide promised.
- TimetableEntry — sprint 2's (term, class, weekday, period, subject, teacher) tuple is the natural anchor for a per-subject session. SubjectAttendanceSession.TimetableEntryId is the only academic foreign key that session needs.
- SchoolClass / Term — the daily register is keyed on (class × date) but also carries TermId so 'all registers for first term' is a single index lookup.

- ILookupService — sprint 4 adds one method (GetAttendanceStatusesAsync). Every other lookup call the attendance pages need (sessions, terms, classes, students) already exists.
- Radzen Blazor + the green/gold app.css — the attendance pages adopt the same .nps-page-header / .nps-card / .nps-form-grid primitives so they read as part of the same product.

1.2 Functional scope delivered

Concretely, after this sprint a SuperAdmin, HeadTeacher, or Teacher signing in to the application can:

1. Open a daily attendance register for any (class × date) combination. Every pupil currently enrolled in that class is pre-loaded with the default status (Present), and the term that covers the date is resolved automatically.
2. Mark or change each pupil's status, set an arrival time when the status is Late, and capture free-text remarks.
3. Submit the register, which locks every entry against further edits until an admin reopens it.
4. Take per-subject attendance: pick a term, class, and date; the page lists every lesson on the timetable for that weekday; click any lesson to open its register; mark each pupil; submit.
5. View a per-class attendance summary across a term — days counted, days present, days late, days absent, days excused, and a colour-coded percentage band per pupil.
6. Soft-delete a register or session that should not have been opened, with a friendly refusal if the register is still in the submitted state.

Parents, Students, Bursars and Storekeepers do not see the Attendance navigation panel; their menu items remain placeholders reserved for later sprints (parent portal in particular will read a child's attendance summary).

1.3 Non-goals deliberately deferred

It is just as important to be explicit about what sprint 4 does NOT do, because every one of these has been weighed and consciously deferred:

- Bell-schedule auto-roll. The current subject-attendance flow still requires a teacher to open each lesson register manually. Auto-creating a session at the start of every period via a background job is a feature-add, not a redesign — the schema supports it today.
- Attendance-driven SMS alerts to parents ("your ward was absent today"). Once the notifications backbone lands (separate sprint), wiring up an event off Submit is a few lines.
- Public holidays / non-school days. The system happily lets you open a register on a Sunday because that is sometimes correct (boarding schools, religious schools); a school-calendar feature deserves its own sprint.
- Excused-absence workflow. Today an admin marks a pupil Excused. A follow-on workflow that captures the parent's request (note from home, doctor's note URL) belongs to the parent portal sprint.
- Bulk import of historical attendance from spreadsheets. Some schools want to backfill the year-to-date register on day one. The schema already supports this; the lift is a service method that loops over rows and calls BulkSetAsync, plus a Razor page with file upload.

- Subject-attendance summary reporting. We have a daily summary; per-subject summaries ("Adaeze missed three Maths lessons this term") are useful but the existing data already supports the query — only the UI is missing.

1.4 Scale of the sprint

By the numbers, this sprint adds:

- 5 new domain entities under src/NaijaPrimeSchool.Domain/Attendance/.
- 4 collection navigations on existing entities (SchoolClass, Term, TimetableEntry, Student).
- 3 DTO files under src/NaijaPrimeSchool.Application/Attendance/Dtos/.
- 2 service contracts under src/NaijaPrimeSchool.Application/Attendance/.
- 2 service implementations under src/NaijaPrimeSchool.Infrastructure/Services/.
- 1 method on ILookupService for the new attendance-status lookup.
- 1 EF Core migration introducing 5 new tables and 13 indexes.
- 1 DatabaseInitializer extension seeding the AttendanceStatus lookup with six rows.
- 3 Razor pages under src/NaijaPrimeSchool.Web/Components/Pages/Attendance/.
- 1 navigation menu addition to surface the new pages.

Everything compiles with zero warnings on .NET 10 (the team-wide warning bar). The code follows the patterns already accepted in sprints 1–3, so the diff is low-friction to review.

2. Design decisions and trade-offs

Before any code was written I made a small number of architectural calls that shaped everything that followed. Re-reading these in isolation makes the code easier to navigate later, and it gives future maintainers permission to revisit choices when the trade-offs change.

2.1 Two parallel attendance models, one lookup

The most consequential decision in this sprint was modelling daily attendance and per-subject attendance as two separate aggregate roots — DailyAttendanceRegister and SubjectAttendanceSession — instead of forcing them into one shape. The trade-offs are real:

- A single 'AttendanceRecord' table with a nullable TimetableEntryId would be tempting on day one, but it would muddle two different keys: (class, date) for daily and (timetable-entry, date) for subject. Indexes for the two shapes would conflict, and the UI would have to filter every list query by 'is daily? is subject?'.
- Two roots keep each table narrow. The unique index on (SchoolClassId, Date) for daily and (TimetableEntryId, Date) for subject is the natural identity for each row, and the indexes are non-overlapping.
- AttendanceStatus is shared between the two. The same Present/Absent/Late/Excused/Sick/Suspended values apply in both models — there is no conceptual reason to split the lookup.
- If a future sprint needs a unified 'show me everything that happened to this pupil today' view, joining the two tables with UNION ALL is cheap. The opposite refactor — splitting one table back into two — would have been much harder.

2.2 No enums — AttendanceStatus is a table

The Present/Absent/Late/Excused/Sick/Suspended set is the textbook argument for a C# enum: it is genuinely closed, semantically clear, and rarely changes. I still modelled it as a table, consistent with sprints 1–3:

- Translatability. A future Hausa or Yoruba translation only needs to edit Name in a row, not recompile.
- Extensibility by data. A school that wants to distinguish between 'Sick (with note)' and 'Sick (no note)' adds one row; no code change.
- Reportability. Joins to the lookup surface readable labels in every reporting tool; magic enum integers do not.
- Display order and rendering hints. AttendanceStatus carries a DisplayOrder for the dropdown ordering and a CountsAsPresent flag the summary code uses to compute percentages without hard-coding which statuses count.
- Code column. Each status carries a short Code (P, L, E, S, A, SP) that the UI can use as a compact visual marker. DailyAttendanceService keys 'late arrival time' off the code value 'L', not the integer index of an enum.

The cost is one extra join on every read query, which EF Core handles efficiently because the lookup table is six rows and effectively always cached.

2.3 Submit / reopen lifecycle, not 'finalised forever'

Both DailyAttendanceRegister and SubjectAttendanceSession carry an IsSubmitted boolean and a nullable SubmittedOn. Once submitted, the service layer refuses every edit until an admin explicitly reopens it. Three considerations went into this:

- Reality. Class teachers submit a register at the end of homeroom; thirty minutes later a child's parent rings to say their car broke down on the school run. The teacher needs to flip the child from Absent to Late. We need to support that without losing the audit trail of who changed what and when.
- Audit trail. The audit columns (CreatedOn/By, ModifiedOn/By) on the register and each entry capture every change. SubmittedOn captures the moment the register was first considered final. ModifiedOn after a reopen makes the post-submission edit visible.
- Soft delete still applies. A submitted register cannot be soft-deleted directly — the service refuses with a friendly message — because deleting submitted attendance is different from amending it. Reopen first, then delete if really required.

2.4 Pre-populate registers from open enrolments

OpenAsync on both services loads every Enrolment for the class with EnrolledOn ≤ Date AND (WithdrawnOn IS NULL OR WithdrawnOn ≥ Date), then creates a default-status entry for each one. The reasoning:

- Teachers should not have to add pupils to the register themselves. The class roster on the date in question is the source of truth.
- A pupil who was withdrawn before the date does not appear. A pupil who is enrolled but was admitted three days after the date does not appear either. The query is the load-bearing definition of 'who is in this class on this day'.
- If a pupil is added or withdrawn after the register is opened, the existing entries are not touched. The register is a snapshot of what the teacher saw that day, not a recomputation of today's roster.

2.5 Daily register carries TermId; subject session does not

There is a deliberate asymmetry between the two roots. DailyAttendanceRegister carries an explicit TermId column. SubjectAttendanceSession does not, because TermId is reachable through TimetableEntry.Term and there is no direct query on session that does not also need to join to the timetable entry anyway.

OpenAsync on DailyAttendanceService resolves the term at creation time using the date and the class's session: first it looks for a term whose start/end range covers the date, and falls back to the current term in the session if no term covers the date (typical for edge-of-holiday days). The chosen term is stamped on the row so subsequent reads do not have to recompute it.

2.6 SubjectAttendanceSession validates date vs weekday

OpenAsync on SubjectAttendanceService refuses to create a session when the calendar weekday of the requested date does not match the WeekDay of the timetable entry. A Monday-period maths lesson on a Wednesday is almost always a typo, and the service catches it before the register is created. The exact check:

```
var weekDayName = entry.WeekDay!.Name;
var actualDayOfWeek = request.Date.DayOfWeek.ToString();
if (!string.Equals(weekDayName, actualDayOfWeek,
                   StringComparison.OrdinalIgnoreCase))
    return OperationResult<Guid>.Failure(
        $"Selected date is a {actualDayOfWeek} but "
      + $"this lesson runs on {weekDayName}.");
```

This relies on the WeekDay.Name ("Monday") matching DayOfWeek.ToString() ("Monday") exactly, which is true for the seeded Monday–Friday rows. If a school adds a Saturday WeekDay row, it just has to be named "Saturday" and the check still works.

2.7 Composite unique indexes

Four composite unique indexes do most of the structural integrity heavy lifting in sprint 4:

- (SchoolClassId, Date) on DailyAttendanceRegister — at most one daily register per class per day.
- (RegisterId, StudentId) on DailyAttendanceEntry — at most one entry per pupil per register.
- (TimetableEntryId, Date) on SubjectAttendanceSession — at most one session per timetable entry per day.
- (SessionId, StudentId) on SubjectAttendanceEntry — at most one entry per pupil per session.

OpenAsync on both services pre-checks the parent-row constraint with .FirstOrDefaultAsync; if a register/session already exists for that key, it returns the existing Id rather than failing. This is what lets a teacher click 'Open register' twice without seeing a database error.

2.8 Foreign-key delete behaviour

- DailyAttendanceEntry.RegisterId / SubjectAttendanceEntry.SessionId → Cascade. If the parent row is hard-deleted, child rows go with it. We cannot reach this state from the UI — the soft-delete flow refuses submitted registers — but the schema is honest about it.
- DailyAttendanceRegister.SchoolClassId / TermId → Restrict. A class or term cannot be hard-deleted while any register points at it. This is consistent with sprint 2's stance on academic structural rows.
- AttendanceStatusId on both entry tables → Restrict. You cannot accidentally lose 'Present' from the lookup while millions of rows point at it.
- TakenById on both parent tables → SetNull. Removing a teacher's account should not delete the registers they took.
- StudentId on both entry tables → Restrict. We never delete a Student row, but if some future maintenance script does, it should fail loudly while attendance entries still point at the pupil.

2.9 Service per attendance model, plus shared lookup

Two services split the work along the natural seams: IDailyAttendanceService and ISubjectAttendanceService. Each one owns OpenAsync, SetEntryAsync, BulkSetAsync, SubmitAsync, ReopenAsync, SoftDeleteAsync, plus the appropriate read methods. IDailyAttendanceService also owns the per-pupil and per-class summary queries because those are sourced from daily attendance — per-subject summaries are deferred to a later sprint.

I considered a single combined IAttendanceService and rejected it: each service is already 200+ lines, and merging them would have made the file unreviewable.

2.10 Inline editing in the grid, not modal forms

Both attendance pages render the entries as a RadzenDataGrid where each cell is the editor. Status is a dropdown bound directly to the row; arrival time and remarks are inline text inputs. Save is a single button that BulkSetAsync's every row in one call. Reasons:

- A class of thirty pupils marked one-by-one through a modal would be cripplingly slow. Inline grid editing is what teachers actually do today on paper.
- BulkSetAsync diffs against the existing entries and updates only the rows that changed, but the API deliberately accepts the full set so the UI does not have to track dirty rows.
- A submit button next to save lets the teacher mark and submit in two clicks: change a few statuses, click 'Save changes' to write the diff, then click 'Submit register' to lock it.

3. The Domain layer in full

Every attendance entity lives in a single new folder, src/NaijaPrimeSchool.Domain/Attendance/. There are no abstract base classes other than BaseEntity, and the entities are deliberately anaemic — no domain methods, no validation logic. Validation lives in the Application DTOs (DataAnnotations) and in the Infrastructure services (cross-aggregate checks). The Domain layer is a typed vocabulary; it is not the place for behaviour in this codebase.

3.1 Folder layout and namespacing

```
src/NaijaPrimeSchool.Domain/
├── Attendance/                        ← (new in sprint 4)
│   ├── AttendanceStatus.cs            ← lookup
│   ├── DailyAttendanceRegister.cs     ← class × date root
│   ├── DailyAttendanceEntry.cs        ← register × pupil
│   ├── SubjectAttendanceSession.cs    ← timetable entry × date
│   └── SubjectAttendanceEntry.cs      ← session × pupil
├── Family/                            ← from sprint 3
├── Academics/                         ← from sprint 2 (3 small additions)
├── Common/                            ← from sprint 1
└── Identity/                          ← from sprint 1
```

3.2 AttendanceStatus.cs — the lookup

The shared lookup. Code is the short two-character marker (P, L, E, S, A, SP); CountsAsPresent gates the percentage calculations.

Listing — src/NaijaPrimeSchool.Domain/Attendance/AttendanceStatus.cs

```
using NaijaPrimeSchool.Domain.Common;

namespace NaijaPrimeSchool.Domain.Attendance;

public class AttendanceStatus : BaseEntity
{
    public string Name { get; set; } = string.Empty;
    public string Code { get; set; } = string.Empty;
    public int DisplayOrder { get; set; }
    public bool CountsAsPresent { get; set; }

    public ICollection<DailyAttendanceEntry> DailyEntries { get; set; } = [];
    public ICollection<SubjectAttendanceEntry> SubjectEntries { get; set; } = [];
}
```

3.3 DailyAttendanceRegister.cs — the daily root

DailyAttendanceRegister is keyed naturally by (SchoolClassId, Date). The TermId is denormalised at creation time to make 'all registers for first term' a single index seek. TakenBy is optional — if a head-teacher takes a class teacher's register on their behalf, that is captured. IsSubmitted + SubmittedOn make up the lifecycle flag.

Listing — src/NaijaPrimeSchool.Domain/Attendance/DailyAttendanceRegister.cs

```
using NaijaPrimeSchool.Domain.Academics;
using NaijaPrimeSchool.Domain.Common;
using NaijaPrimeSchool.Domain.Identity;

namespace NaijaPrimeSchool.Domain.Attendance;

public class DailyAttendanceRegister : BaseEntity
{
    public Guid SchoolClassId { get; set; }
    public SchoolClass? SchoolClass { get; set; }

    public Guid TermId { get; set; }
    public Term? Term { get; set; }

    public DateOnly Date { get; set; }

    public Guid? TakenById { get; set; }
    public ApplicationUser? TakenBy { get; set; }
    public DateTimeOffset? TakenOn { get; set; }

    public bool IsSubmitted { get; set; }
    public DateTimeOffset? SubmittedOn { get; set; }

    public string? Notes { get; set; }

    public ICollection<DailyAttendanceEntry> Entries { get; set; } = [];
}
```

3.4 DailyAttendanceEntry.cs — one row per pupil

Each entry points at a Student and an AttendanceStatus. ArrivalTime is optional and only meaningful when the status is Late; the UI hides the field for other statuses. Remarks is free-text for any unusual case.

Listing — src/NaijaPrimeSchool.Domain/Attendance/DailyAttendanceEntry.cs

```
using NaijaPrimeSchool.Domain.Common;
using NaijaPrimeSchool.Domain.Family;

namespace NaijaPrimeSchool.Domain.Attendance;

public class DailyAttendanceEntry : BaseEntity
{
    public Guid RegisterId { get; set; }
    public DailyAttendanceRegister? Register { get; set; }

    public Guid StudentId { get; set; }
    public Student? Student { get; set; }

    public Guid AttendanceStatusId { get; set; }
    public AttendanceStatus? AttendanceStatus { get; set; }

    public TimeOnly? ArrivalTime { get; set; }
    public string? Remarks { get; set; }
}
```

3.5 SubjectAttendanceSession.cs — the per-lesson root

SubjectAttendanceSession hangs off TimetableEntry — every concept the academic side already knows about (term, class, subject, weekday, period, teacher, room) is reached through that single navigation. The session itself only adds a Date, a TakenBy, the lifecycle flags, and an optional Notes column.

Listing — src/NaijaPrimeSchool.Domain/Attendance/SubjectAttendanceSession.cs

```
using NaijaPrimeSchool.Domain.Academics;
using NaijaPrimeSchool.Domain.Common;
using NaijaPrimeSchool.Domain.Identity;

namespace NaijaPrimeSchool.Domain.Attendance;

public class SubjectAttendanceSession : BaseEntity
{
    public Guid TimetableEntryId { get; set; }
    public TimetableEntry? TimetableEntry { get; set; }

    public DateOnly Date { get; set; }

    public Guid? TakenById { get; set; }
    public ApplicationUser? TakenBy { get; set; }
    public DateTimeOffset? TakenOn { get; set; }

    public bool IsSubmitted { get; set; }
    public DateTimeOffset? SubmittedOn { get; set; }

    public string? Notes { get; set; }

    public ICollection<SubjectAttendanceEntry> Entries { get; set; } = [];
}
```

3.6 SubjectAttendanceEntry.cs

Mirrors DailyAttendanceEntry but without the ArrivalTime column — for a 40-minute lesson, 'arrival time' is not a meaningful concept. Status + remarks is enough.

Listing — src/NaijaPrimeSchool.Domain/Attendance/SubjectAttendanceEntry.cs

```
using NaijaPrimeSchool.Domain.Common;
using NaijaPrimeSchool.Domain.Family;

namespace NaijaPrimeSchool.Domain.Attendance;

public class SubjectAttendanceEntry : BaseEntity
{
    public Guid SessionId { get; set; }
    public SubjectAttendanceSession? Session { get; set; }

    public Guid StudentId { get; set; }
    public Student? Student { get; set; }
```

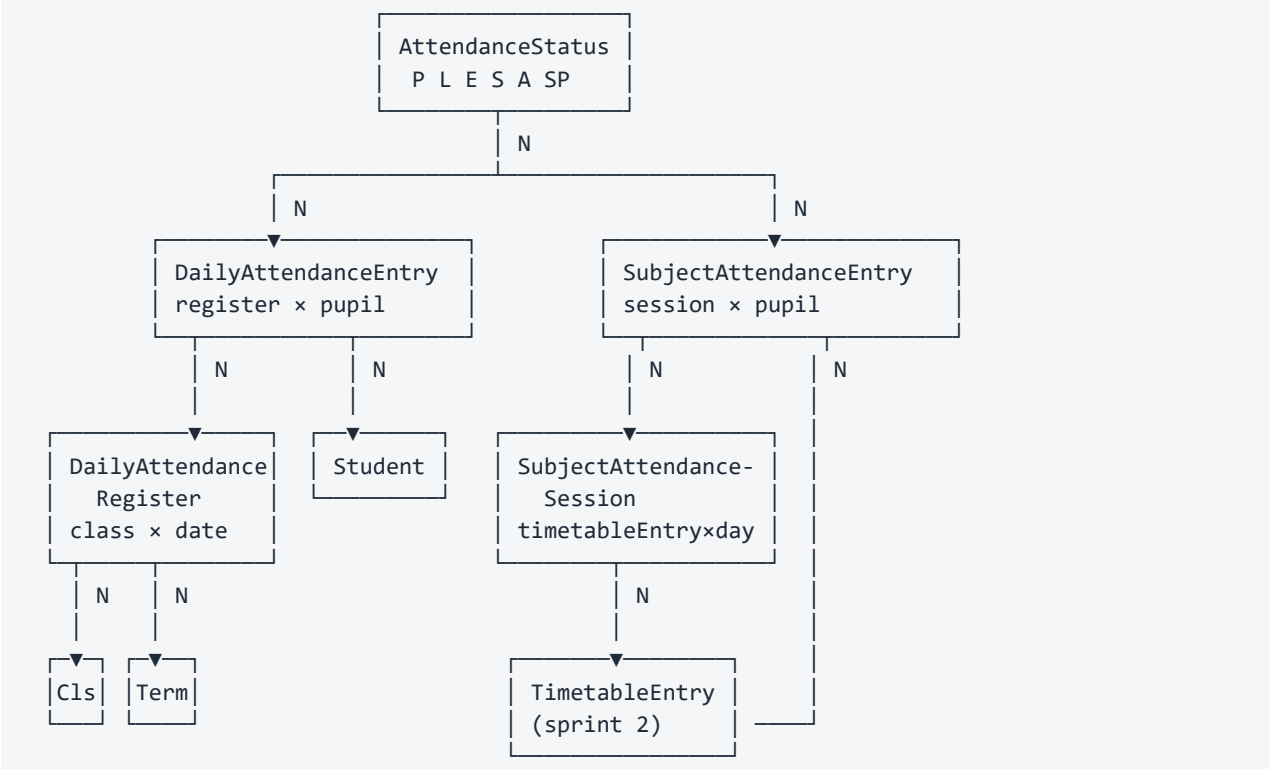
```
    public Guid AttendanceStatusId { get; set; }
    public AttendanceStatus? AttendanceStatus { get; set; }

    public string? Remarks { get; set; }
}
```

3.7 Existing entities that grew a navigation

Four sprint 1–3 entities pick up exactly one collection navigation each. Existing code that ignored these properties continues to compile and behave identically; the migration emits no schema change to the host tables.

- SchoolClass.DailyAttendanceRegisters
- Term.DailyAttendanceRegisters
- TimetableEntry.AttendanceSessions
- Student.DailyAttendanceEntries and Student.SubjectAttendanceEntries

3.8 Relationships at a glance



4. Application layer — DTOs and contracts

Application is the boundary between the Razor UI and the persistence layer. It is deliberately thin: just DTOs and service interfaces. Sprint 4 adds two new subfolders: Attendance/Dtos and Attendance.

4.1 DTO design rules

- Each list-view DTO carries denormalised display fields (SchoolClassName, SubjectName, TermName) so the UI can render a row without lazy-loading any navigations.
- Detail DTOs (DailyAttendanceRegisterDetailDto, SubjectAttendanceSessionDetailDto) bundle the parent row and its entries into one payload, so the editor page is a single round-trip.
- Bulk-set requests carry the full set of entries; the service diffs against existing rows. The UI does not have to track dirty rows.
- Filter types (DailyRegisterListFilter, SubjectSessionListFilter, StudentAttendanceSummaryFilter) carry the search/scope parameters.

4.2 DailyAttendanceDtos.cs

Listing — src/NaijaPrimeSchool.Application/Attendance/Dtos/DailyAttendanceDtos.cs

```
using System.ComponentModel.DataAnnotations;

namespace NaijaPrimeSchool.Application.Attendance.Dtos;

public class DailyAttendanceRegisterDto
{
    public Guid Id { get; set; }

    public Guid SchoolClassId { get; set; }
    public string SchoolClassName { get; set; } = string.Empty;

    public Guid SessionId { get; set; }
    public string SessionName { get; set; } = string.Empty;

    public Guid TermId { get; set; }
    public string TermName { get; set; } = string.Empty;

    public DateOnly Date { get; set; }

    public Guid? TakenById { get; set; }
    public string? TakenByName { get; set; }
    public DateTimeOffset? TakenOn { get; set; }

    public bool IsSubmitted { get; set; }
    public DateTimeOffset? SubmittedOn { get; set; }

    public string? Notes { get; set; }

    public int PresentCount { get; set; }
    public int AbsentCount { get; set; }
    public int LateCount { get; set; }
    public int TotalCount { get; set; }
}
```

```
public class DailyAttendanceEntryDto
{
    public Guid Id { get; set; }
    public Guid RegisterId { get; set; }

    public Guid StudentId { get; set; }
    public string StudentName { get; set; } = string.Empty;
    public string StudentAdmissionNumber { get; set; } = string.Empty;

    public Guid AttendanceStatusId { get; set; }
    public string AttendanceStatusName { get; set; } = string.Empty;
    public string AttendanceStatusCode { get; set; } = string.Empty;
    public bool CountsAsPresent { get; set; }

    public TimeOnly? ArrivalTime { get; set; }
    public string? Remarks { get; set; }
}

public class DailyAttendanceRegisterDetailDto
{
    public DailyAttendanceRegisterDto Register { get; set; } = new();
    public List<DailyAttendanceEntryDto> Entries { get; set; } = [];
}

public class OpenDailyRegisterRequest
{
    [Required] public Guid SchoolClassId { get; set; }
    [Required] public DateOnly Date { get; set; }
    public Guid? TakenById { get; set; }
}

public class UpsertDailyEntryRequest
{
    public Guid? Id { get; set; }
    [Required] public Guid RegisterId { get; set; }
    [Required] public Guid StudentId { get; set; }
    [Required] public Guid AttendanceStatusId { get; set; }
    public TimeOnly? ArrivalTime { get; set; }

    [StringLength(300)]
    public string? Remarks { get; set; }
}

public class BulkSetDailyAttendanceRequest
{
    [Required] public Guid RegisterId { get; set; }
    public List<UpsertDailyEntryRequest> Entries { get; set; } = [];
}

public class DailyRegisterListFilter
{
    public Guid? SchoolClassId { get; set; }
    public Guid? TermId { get; set; }
```

```
    public Guid? SessionId { get; set; }
    public DateOnly? FromDate { get; set; }
    public DateOnly? ToDate { get; set; }
    public bool? IsSubmitted { get; set; }
}
```

PresentCount, AbsentCount, LateCount and TotalCount are populated by the service projection so the list view can show 'P:28 A:1 L:1 / 30' for each register without a follow-up query.

4.3 SubjectAttendanceDtos.cs

Listing — src/NaijaPrimeSchool.Application/Attendance/Dtos/SubjectAttendanceDtos.cs

```
using System.ComponentModel.DataAnnotations;

namespace NaijaPrimeSchool.Application.Attendance.Dtos;

public class SubjectAttendanceSessionDto
{
    public Guid Id { get; set; }

    public Guid TimetableEntryId { get; set; }

    public Guid SchoolClassId { get; set; }
    public string SchoolClassName { get; set; } = string.Empty;

    public Guid SubjectId { get; set; }
    public string SubjectName { get; set; } = string.Empty;
    public string SubjectCode { get; set; } = string.Empty;

    public Guid TermId { get; set; }
    public string TermName { get; set; } = string.Empty;

    public Guid SessionId { get; set; }
    public string SessionName { get; set; } = string.Empty;

    public Guid WeekDayId { get; set; }
    public string WeekDayName { get; set; } = string.Empty;

    public Guid TimetablePeriodId { get; set; }
    public string PeriodName { get; set; } = string.Empty;
    public TimeOnly PeriodStart { get; set; }
    public TimeOnly PeriodEnd { get; set; }

    public DateOnly Date { get; set; }

    public Guid? TakenById { get; set; }
    public string? TakenByName { get; set; }
    public DateTimeOffset? TakenOn { get; set; }

    public bool IsSubmitted { get; set; }
    public DateTimeOffset? SubmittedOn { get; set; }

    public string? Notes { get; set; }
```

```
    public int PresentCount { get; set; }
    public int AbsentCount { get; set; }
    public int TotalCount { get; set; }
}

public class SubjectAttendanceEntryDto
{
    public Guid Id { get; set; }
    public Guid SessionId { get; set; }

    public Guid StudentId { get; set; }
    public string StudentName { get; set; } = string.Empty;
    public string StudentAdmissionNumber { get; set; } = string.Empty;

    public Guid AttendanceStatusId { get; set; }
    public string AttendanceStatusName { get; set; } = string.Empty;
    public string AttendanceStatusCode { get; set; } = string.Empty;
    public bool CountsAsPresent { get; set; }

    public string? Remarks { get; set; }
}

public class SubjectAttendanceSessionDetailDto
{
    public SubjectAttendanceSessionDto Session { get; set; } = new();
    public List<SubjectAttendanceEntryDto> Entries { get; set; } = [];
}

public class OpenSubjectSessionRequest
{
    [Required] public Guid TimetableEntryId { get; set; }
    [Required] public DateOnly Date { get; set; }
    public Guid? TakenById { get; set; }
}

public class UpsertSubjectEntryRequest
{
    public Guid? Id { get; set; }
    [Required] public Guid SessionId { get; set; }
    [Required] public Guid StudentId { get; set; }
    [Required] public Guid AttendanceStatusId { get; set; }

    [StringLength(300)]
    public string? Remarks { get; set; }
}

public class BulkSetSubjectAttendanceRequest
{
    [Required] public Guid SessionId { get; set; }
    public List<UpsertSubjectEntryRequest> Entries { get; set; } = [];
}

public class SubjectSessionListFilter
```

```
{
    public Guid? TimetableEntryId { get; set; }
    public Guid? SchoolClassId { get; set; }
    public Guid? TermId { get; set; }
    public Guid? SubjectId { get; set; }
    public DateOnly? FromDate { get; set; }
    public DateOnly? ToDate { get; set; }
}
```

4.4 AttendanceSummaryDtos.cs

Listing — src/NaijaPrimeSchool.Application/Attendance/Dtos/AttendanceSummaryDtos.cs

```
namespace NaijaPrimeSchool.Application.Attendance.Dtos;

public class StudentAttendanceSummaryDto
{
    public Guid StudentId { get; set; }
    public string StudentName { get; set; } = string.Empty;
    public string StudentAdmissionNumber { get; set; } = string.Empty;

    public int DaysCounted { get; set; }
    public int DaysPresent { get; set; }
    public int DaysAbsent { get; set; }
    public int DaysLate { get; set; }
    public int DaysExcused { get; set; }

    public double PresentRate =>
        DaysCounted == 0 ? 0d : Math.Round(DaysPresent * 100d / DaysCounted, 1);
}

public class StudentAttendanceSummaryFilter
{
    public Guid StudentId { get; set; }
    public Guid? TermId { get; set; }
    public Guid? SessionId { get; set; }
    public DateOnly? FromDate { get; set; }
    public DateOnly? ToDate { get; set; }
}

public class ClassAttendanceSummaryDto
{
    public Guid SchoolClassId { get; set; }
    public string SchoolClassName { get; set; } = string.Empty;
    public List<StudentAttendanceSummaryDto> Students { get; set; } = [];
}
```

PresentRate is a computed property in the DTO itself — rounded to one decimal place, returns 0 when DaysCounted is zero. The summary grid uses this directly to colour the badge (≥ 90% green, ≥ 75% amber, otherwise red).

4.5 IDailyAttendanceService.cs

Listing — src/NaijaPrimeSchool.Application/Attendance/IDailyAttendanceService.cs

```
using NaijaPrimeSchool.Application.Attendance.Dtos;
using NaijaPrimeSchool.Application.Common;

namespace NaijaPrimeSchool.Application.Attendance;

public interface IDailyAttendanceService
{
    Task<IReadOnlyList<DailyAttendanceRegisterDto>> ListAsync(DailyRegisterListFilter
filter, CancellationToken ct = default);
    Task<DailyAttendanceRegisterDetailDto?> GetByIdAsync(Guid id, CancellationToken ct =
default);
    Task<DailyAttendanceRegisterDetailDto?> GetForClassDateAsync(Guid schoolClassId,
DateOnly date, CancellationToken ct = default);

    Task<OperationResult<Guid>> OpenAsync(OpenDailyRegisterRequest request,
CancellationToken ct = default);
    Task<OperationResult> SetEntryAsync(UpsertDailyEntryRequest request, CancellationToken
ct = default);
    Task<OperationResult> BulkSetAsync(BulkSetDailyAttendanceRequest request,
CancellationToken ct = default);
    Task<OperationResult> SubmitAsync(Guid registerId, CancellationToken ct = default);
    Task<OperationResult> ReopenAsync(Guid registerId, CancellationToken ct = default);
    Task<OperationResult> SoftDeleteAsync(Guid registerId, CancellationToken ct = default);

    Task<StudentAttendanceSummaryDto> GetStudentSummaryAsync(StudentAttendanceSummaryFilter
filter, CancellationToken ct = default);
    Task<ClassAttendanceSummaryDto> GetClassSummaryAsync(Guid schoolClassId, Guid? termId,
CancellationToken ct = default);
}
```

4.6 ISubjectAttendanceService.cs

Listing — src/NaijaPrimeSchool.Application/Attendance/ISubjectAttendanceService.cs

```
using NaijaPrimeSchool.Application.Attendance.Dtos;
using NaijaPrimeSchool.Application.Common;

namespace NaijaPrimeSchool.Application.Attendance;

public interface ISubjectAttendanceService
{
    Task<IReadOnlyList<SubjectAttendanceSessionDto>> ListAsync(SubjectSessionListFilter
filter, CancellationToken ct = default);
    Task<SubjectAttendanceSessionDetailDto?> GetByIdAsync(Guid id, CancellationToken ct =
default);
    Task<SubjectAttendanceSessionDetailDto?> GetForEntryDateAsync(Guid timetableEntryId,
DateOnly date, CancellationToken ct = default);

    Task<OperationResult<Guid>> OpenAsync(OpenSubjectSessionRequest request,
CancellationToken ct = default);
    Task<OperationResult> SetEntryAsync(UpsertSubjectEntryRequest request,
```

```
CancellationToken ct = default);
    Task<OperationResult> BulkSetAsync(BulkSetSubjectAttendanceRequest request,
CancellationToken ct = default);
    Task<OperationResult> SubmitAsync(Guid sessionId, CancellationToken ct = default);
    Task<OperationResult> ReopenAsync(Guid sessionId, CancellationToken ct = default);
    Task<OperationResult> SoftDeleteAsync(Guid sessionId, CancellationToken ct = default);
}
```

4.7 ILookupService extension

ILookupService grew by one method, GetAttendanceStatusesAsync, which surfaces the AttendanceStatus lookup as a list of LookupDto items (Id, Name, Code). The Code is what the UI uses to spot 'Late' specifically when deciding whether to show the arrival-time field.

Excerpt — ILookupService.cs (sprint 4 addition)

```
    Task<IReadOnlyList<LookupDto>> GetAttendanceStatusesAsync(CancellationToken ct =
default);
```

5. Infrastructure — DbContext changes

ApplicationDbContext picks up five new DbSets and a single new ConfigureAttendance method. Everything else in the file is unchanged from sprint 3. The pattern of a private ConfigureXxx method per feature folder keeps OnModelCreating lean.

5.1 New DbSets

Excerpt — the five attendance DbSets

```
    public DbSet<AttendanceStatus> AttendanceStatuses => Set<AttendanceStatus>();
    public DbSet<DailyAttendanceRegister> DailyAttendanceRegisters =>
Set<DailyAttendanceRegister>();
    public DbSet<DailyAttendanceEntry> DailyAttendanceEntries =>
Set<DailyAttendanceEntry>();
    public DbSet<SubjectAttendanceSession> SubjectAttendanceSessions =>
Set<SubjectAttendanceSession>();
    public DbSet<SubjectAttendanceEntry> SubjectAttendanceEntries =>
Set<SubjectAttendanceEntry>();

    protected override void OnModelCreating(ModelBuilder builder)
```

5.2 ConfigureAttendance

ConfigureAttendance is invoked from OnModelCreating after ConfigureFamily. It uses the same ConfigureLookup helper from sprint 1 for AttendanceStatus, then configures the two register/entry pairs in turn.

Excerpt — ConfigureAttendance

```
    private static void ConfigureAttendance(ModelBuilder builder)
    {
        ConfigureLookup<AttendanceStatus>(builder, "AttendanceStatuses", extra: b =>
        {
            b.Property(s => s.Name).HasMaxLength(40).IsRequired();
            b.Property(s => s.Code).HasMaxLength(5).IsRequired();
            b.HasIndex(s => s.Name).IsUnique();
            b.HasIndex(s => s.Code).IsUnique();
        });

        builder.Entity<DailyAttendanceRegister>(b =>
        {
            b.ToTable("DailyAttendanceRegisters");
            b.HasKey(r => r.Id);
            b.Property(r => r.Notes).HasMaxLength(500);
            b.Property(r => r.CreatedBy).HasMaxLength(100);
            b.Property(r => r.ModifiedBy).HasMaxLength(100);
            b.Property(r => r.DeletedBy).HasMaxLength(100);

            b.HasOne(r => r.SchoolClass).WithMany(c => c.DailyAttendanceRegisters)
                .HasForeignKey(r => r.SchoolClassId)
                .OnDelete(DeleteBehavior.Restrict);

            b.HasOne(r => r.Term).WithMany(t => t.DailyAttendanceRegisters)
```

```
            .HasForeignKey(r => r.TermId)
            .OnDelete(DeleteBehavior.Restrict);

        b.HasOne(r => r.TakenBy).WithMany()
            .HasForeignKey(r => r.TakenById)
            .OnDelete(DeleteBehavior.SetNull);

        // One register per (class, date).
        b.HasIndex(r => new { r.SchoolClassId, r.Date }).IsUnique();
        b.HasIndex(r => r.Date);
        b.HasIndex(r => r.IsDeleted);
        b.HasQueryFilter(r => !r.IsDeleted);
    });

    builder.Entity<DailyAttendanceEntry>(b =>
    {
        b.ToTable("DailyAttendanceEntries");
        b.HasKey(e => e.Id);
        b.Property(e => e.Remarks).HasMaxLength(300);
        b.Property(e => e.CreatedBy).HasMaxLength(100);
        b.Property(e => e.ModifiedBy).HasMaxLength(100);
        b.Property(e => e.DeletedBy).HasMaxLength(100);

        b.HasOne(e => e.Register).WithMany(r => r.Entries)
            .HasForeignKey(e => e.RegisterId)
            .OnDelete(DeleteBehavior.Cascade);

        b.HasOne(e => e.Student).WithMany(s => s.DailyAttendanceEntries)
            .HasForeignKey(e => e.StudentId)
            .OnDelete(DeleteBehavior.Restrict);

        b.HasOne(e => e.AttendanceStatus).WithMany(s => s.DailyEntries)
            .HasForeignKey(e => e.AttendanceStatusId)
            .OnDelete(DeleteBehavior.Restrict);

        // One entry per (register, student).
        b.HasIndex(e => new { e.RegisterId, e.StudentId }).IsUnique();
        b.HasIndex(e => e.IsDeleted);
        b.HasQueryFilter(e => !e.IsDeleted);
    });

    builder.Entity<SubjectAttendanceSession>(b =>
    {
        b.ToTable("SubjectAttendanceSessions");
        b.HasKey(s => s.Id);
        b.Property(s => s.Notes).HasMaxLength(500);
        b.Property(s => s.CreatedBy).HasMaxLength(100);
        b.Property(s => s.ModifiedBy).HasMaxLength(100);
        b.Property(s => s.DeletedBy).HasMaxLength(100);

        b.HasOne(s => s.TimetableEntry).WithMany(e => e.AttendanceSessions)
            .HasForeignKey(s => s.TimetableEntryId)
            .OnDelete(DeleteBehavior.Restrict);
```

```
            b.HasOne(s => s.TakenBy).WithMany()
                .HasForeignKey(s => s.TakenById)
                .OnDelete(DeleteBehavior.SetNull);

            // One session per (timetable entry, date).
            b.HasIndex(s => new { s.TimetableEntryId, s.Date }).IsUnique();
            b.HasIndex(s => s.Date);
            b.HasIndex(s => s.IsDeleted);
            b.HasQueryFilter(s => !s.IsDeleted);
        });

        builder.Entity<SubjectAttendanceEntry>(b =>
        {
            b.ToTable("SubjectAttendanceEntries");
            b.HasKey(e => e.Id);
            b.Property(e => e.Remarks).HasMaxLength(300);
            b.Property(e => e.CreatedBy).HasMaxLength(100);
            b.Property(e => e.ModifiedBy).HasMaxLength(100);
            b.Property(e => e.DeletedBy).HasMaxLength(100);

            b.HasOne(e => e.Session).WithMany(s => s.Entries)
                .HasForeignKey(e => e.SessionId)
                .OnDelete(DeleteBehavior.Cascade);

            b.HasOne(e => e.Student).WithMany(s => s.SubjectAttendanceEntries)
                .HasForeignKey(e => e.StudentId)
                .OnDelete(DeleteBehavior.Restrict);

            b.HasOne(e => e.AttendanceStatus).WithMany(s => s.SubjectEntries)
                .HasForeignKey(e => e.AttendanceStatusId)
                .OnDelete(DeleteBehavior.Restrict);

            // One entry per (session, student).
            b.HasIndex(e => new { e.SessionId, e.StudentId }).IsUnique();
            b.HasIndex(e => e.IsDeleted);
            b.HasQueryFilter(e => !e.IsDeleted);
        });
    }

    private static void ConfigureLookup<TEntity>(
```

5.3 SaveChanges still does all the work

The SaveChanges override from sprint 1 needs no changes for sprint 4. Every new entity inherits BaseEntity, so the audit stamping and soft-delete rewriting apply automatically. The pattern continues to pay off.

6. Infrastructure — service implementations

Two new services land in src/NaijaPrimeSchool.Infrastructure/Services/. Each one is a straightforward implementation of the matching Application interface. LookupService grows by one method.

6.1 DailyAttendanceService.cs

DailyAttendanceService is the heavier of the two. It owns the daily-register CRUD, the bulk update path, the submit/reopen lifecycle, and the per-student / per-class summary queries.

Listing — src/NaijaPrimeSchool.Infrastructure/Services/DailyAttendanceService.cs

```
using Microsoft.EntityFrameworkCore;
using NaijaPrimeSchool.Application.Attendance;
using NaijaPrimeSchool.Application.Attendance.Dtos;
using NaijaPrimeSchool.Application.Common;
using NaijaPrimeSchool.Domain.Attendance;
using NaijaPrimeSchool.Domain.Family;
using NaijaPrimeSchool.Infrastructure.Persistence;

namespace NaijaPrimeSchool.Infrastructure.Services;

public class DailyAttendanceService(ApplicationDbContext db) : IDailyAttendanceService
{
    private static IQueryable<DailyAttendanceRegisterDto>
ProjectRegister(IQueryable<DailyAttendanceRegister> q) =>
        q.Select(r => new DailyAttendanceRegisterDto
        {
            Id = r.Id,
            SchoolClassId = r.SchoolClassId,
            SchoolClassName = r.SchoolClass!.Name,
            SessionId = r.SchoolClass!.SessionId,
            SessionName = r.SchoolClass!.Session!.Name,
            TermId = r.TermId,
            TermName = r.Term!.TermType!.Name,
            Date = r.Date,
            TakenById = r.TakenById,
            TakenByName = r.TakenBy == null
                ? null
                : (r.TakenBy.FirstName + " " + r.TakenBy.LastName).Trim(),
            TakenOn = r.TakenOn,
            IsSubmitted = r.IsSubmitted,
            SubmittedOn = r.SubmittedOn,
            Notes = r.Notes,
            PresentCount = r.Entries.Count(e => e.AttendanceStatus!.CountsAsPresent),
            AbsentCount = r.Entries.Count(e => !e.AttendanceStatus!.CountsAsPresent),
            LateCount = r.Entries.Count(e => e.AttendanceStatus!.Code == "L"),
            TotalCount = r.Entries.Count,
        });

    public async Task<IReadOnlyList<DailyAttendanceRegisterDto>>
ListAsync(DailyRegisterListFilter filter, CancellationToken ct = default)
    {
        var q = db.DailyAttendanceRegisters.AsQueryable();
```

```
        if (filter.SchoolClassId.HasValue)
            q = q.Where(r => r.SchoolClassId == filter.SchoolClassId.Value);
        if (filter.TermId.HasValue)
            q = q.Where(r => r.TermId == filter.TermId.Value);
        if (filter.SessionId.HasValue)
            q = q.Where(r => r.SchoolClass!.SessionId == filter.SessionId.Value);
        if (filter.FromDate.HasValue)
            q = q.Where(r => r.Date >= filter.FromDate.Value);
        if (filter.ToDate.HasValue)
            q = q.Where(r => r.Date <= filter.ToDate.Value);
        if (filter.IsSubmitted.HasValue)
            q = q.Where(r => r.IsSubmitted == filter.IsSubmitted.Value);

        return await ProjectRegister(q.OrderByDescending(r => r.Date))
            .ToListAsync(ct);
    }

    public async Task<DailyAttendanceRegisterDetailDto?> GetByIdAsync(Guid id,
CancellationToken ct = default)
    {
        var register = await ProjectRegister(db.DailyAttendanceRegisters.Where(r => r.Id ==
id))
            .FirstOrDefaultAsync(ct);
        if (register is null) return null;

        var entries = await ProjectEntries(db.DailyAttendanceEntries.Where(e =>
e.RegisterId == id))
            .ToListAsync(ct);

        return new DailyAttendanceRegisterDetailDto { Register = register, Entries =
entries };
    }

    public async Task<DailyAttendanceRegisterDetailDto?> GetForClassDateAsync(Guid
schoolClassId, DateOnly date, CancellationToken ct = default)
    {
        var register = await db.DailyAttendanceRegisters
            .Where(r => r.SchoolClassId == schoolClassId && r.Date == date)
            .Select(r => r.Id)
            .FirstOrDefaultAsync(ct);

        if (register == Guid.Empty) return null;
        return await GetByIdAsync(register, ct);
    }

    public async Task<OperationResult<Guid>> OpenAsync(OpenDailyRegisterRequest request,
CancellationToken ct = default)
    {
        var schoolClass = await db.SchoolClasses
            .Include(c => c.Session)
            .FirstOrDefaultAsync(c => c.Id == request.SchoolClassId, ct);
        if (schoolClass is null)
            return OperationResult<Guid>.Failure("Class not found.");
```

```
        var term = await db.Terms
            .Where(t => t.SessionId == schoolClass.SessionId
                        && t.StartDate <= request.Date
                        && t.EndDate >= request.Date)
            .FirstOrDefaultAsync(ct);

        if (term is null)
        {
            term = await db.Terms
                .Where(t => t.SessionId == schoolClass.SessionId && t.IsCurrent)
                .FirstOrDefaultAsync(ct);
        }

        if (term is null)
            return OperationResult<Guid>.Failure(
                "No term covers this date and no current term is set for the class's
session.");

        var existing = await db.DailyAttendanceRegisters
            .FirstOrDefaultAsync(r => r.SchoolClassId == request.SchoolClassId && r.Date ==
request.Date, ct);
        if (existing is not null)
            return OperationResult<Guid>.Success(existing.Id);

        var register = new DailyAttendanceRegister
        {
            SchoolClassId = request.SchoolClassId,
            TermId = term.Id,
            Date = request.Date,
            TakenById = request.TakenById,
            TakenOn = request.TakenById.HasValue ? DateTimeOffset.UtcNow : null,
            IsSubmitted = false,
        };
        db.DailyAttendanceRegisters.Add(register);

        var defaultStatus = await db.AttendanceStatuses
            .OrderBy(s => s.DisplayOrder)
            .FirstOrDefaultAsync(ct);
        if (defaultStatus is null)
            return OperationResult<Guid>.Failure("Attendance statuses are not seeded.");

        var enrolledStudents = await db.Enrolments
            .Where(e => e.SchoolClassId == request.SchoolClassId
                        && e.EnrolledOn <= request.Date
                        && (e.WithdrawnOn == null || e.WithdrawnOn >= request.Date))
            .Select(e => e.StudentId)
            .ToListAsync(ct);

        foreach (var studentId in enrolledStudents)
        {
            db.DailyAttendanceEntries.Add(new DailyAttendanceEntry
            {
                Register = register,
                StudentId = studentId,
```

```
                AttendanceStatusId = defaultStatus.Id,
            });
        }

        await db.SaveChangesAsync(ct);
        return OperationResult<Guid>.Success(register.Id);
    }

    public async Task<OperationResult> SetEntryAsync(UpsertDailyEntryRequest request,
CancellationToken ct = default)
    {
        var register = await db.DailyAttendanceRegisters.FirstOrDefaultAsync(r => r.Id ==
request.RegisterId, ct);
        if (register is null) return OperationResult.Failure("Register not found.");
        if (register.IsSubmitted) return OperationResult.Failure("Register is submitted.
Reopen it first.");

        if (!await db.AttendanceStatuses.AnyAsync(s => s.Id == request.AttendanceStatusId,
ct))
            return OperationResult.Failure("Attendance status not valid.");

        if (!await db.Students.AnyAsync(s => s.Id == request.StudentId, ct))
            return OperationResult.Failure("Student not found.");

        var entry = await db.DailyAttendanceEntries
            .FirstOrDefaultAsync(e => e.RegisterId == request.RegisterId && e.StudentId ==
request.StudentId, ct);

        if (entry is null)
        {
            entry = new DailyAttendanceEntry
            {
                RegisterId = request.RegisterId,
                StudentId = request.StudentId,
                AttendanceStatusId = request.AttendanceStatusId,
                ArrivalTime = request.ArrivalTime,
                Remarks = request.Remarks,
            };
            db.DailyAttendanceEntries.Add(entry);
        }
        else
        {
            entry.AttendanceStatusId = request.AttendanceStatusId;
            entry.ArrivalTime = request.ArrivalTime;
            entry.Remarks = request.Remarks;
        }

        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    public async Task<OperationResult> BulkSetAsync(BulkSetDailyAttendanceRequest request,
CancellationToken ct = default)
    {
```

```
        var register = await db.DailyAttendanceRegisters.FirstOrDefaultAsync(r => r.Id ==
request.RegisterId, ct);
        if (register is null) return OperationResult.Failure("Register not found.");
        if (register.IsSubmitted) return OperationResult.Failure("Register is submitted.
Reopen it first.");

        var validStatusIds = await db.AttendanceStatuses.Select(s => s.Id).ToListAsync(ct);
        var statusSet = validStatusIds.ToHashSet();

        var existing = await db.DailyAttendanceEntries
            .Where(e => e.RegisterId == request.RegisterId)
            .ToDictionaryAsync(e => e.StudentId, ct);

        foreach (var item in request.Entries)
        {
            if (!statusSet.Contains(item.AttendanceStatusId))
                return OperationResult.Failure("One or more attendance statuses are
invalid.");

            if (existing.TryGetValue(item.StudentId, out var entry))
            {
                entry.AttendanceStatusId = item.AttendanceStatusId;
                entry.ArrivalTime = item.ArrivalTime;
                entry.Remarks = item.Remarks;
            }
            else
            {
                db.DailyAttendanceEntries.Add(new DailyAttendanceEntry
                {
                    RegisterId = request.RegisterId,
                    StudentId = item.StudentId,
                    AttendanceStatusId = item.AttendanceStatusId,
                    ArrivalTime = item.ArrivalTime,
                    Remarks = item.Remarks,
                });
            }
        }

        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    public async Task<OperationResult> SubmitAsync(Guid registerId, CancellationToken ct =
default)
    {
        var register = await db.DailyAttendanceRegisters.FirstOrDefaultAsync(r => r.Id ==
registerId, ct);
        if (register is null) return OperationResult.Failure("Register not found.");
        if (register.IsSubmitted) return OperationResult.Success();

        register.IsSubmitted = true;
        register.SubmittedOn = DateTimeOffset.UtcNow;
        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
```

```
    }

    public async Task<OperationResult> ReopenAsync(Guid registerId, CancellationToken ct =
default)
    {
        var register = await db.DailyAttendanceRegisters.FirstOrDefaultAsync(r => r.Id ==
registerId, ct);
        if (register is null) return OperationResult.Failure("Register not found.");

        register.IsSubmitted = false;
        register.SubmittedOn = null;
        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    public async Task<OperationResult> SoftDeleteAsync(Guid registerId, CancellationToken
ct = default)
    {
        var register = await db.DailyAttendanceRegisters.FirstOrDefaultAsync(r => r.Id ==
registerId, ct);
        if (register is null) return OperationResult.Failure("Register not found.");

        if (register.IsSubmitted)
            return OperationResult.Failure("Submitted registers cannot be deleted. Reopen
first.");

        db.DailyAttendanceRegisters.Remove(register);
        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    public async Task<StudentAttendanceSummaryDto>
GetStudentSummaryAsync(StudentAttendanceSummaryFilter filter, CancellationToken ct =
default)
    {
        var student = await db.Students.Where(s => s.Id == filter.StudentId)
            .Select(s => new { s.FirstName, s.LastName, s.AdmissionNumber })
            .FirstOrDefaultAsync(ct);

        var query = db.DailyAttendanceEntries
            .Where(e => e.StudentId == filter.StudentId);

        if (filter.TermId.HasValue)
            query = query.Where(e => e.Register!.TermId == filter.TermId.Value);
        if (filter.SessionId.HasValue)
            query = query.Where(e => e.Register!.Term!.SessionId ==
filter.SessionId.Value);
        if (filter.FromDate.HasValue)
            query = query.Where(e => e.Register!.Date >= filter.FromDate.Value);
        if (filter.ToDate.HasValue)
            query = query.Where(e => e.Register!.Date <= filter.ToDate.Value);

        var counts = await query
            .GroupBy(_ => 1)
```

```
            .Select(g => new
            {
                DaysCounted = g.Count(),
                DaysPresent = g.Count(e => e.AttendanceStatus!.CountsAsPresent),
                DaysLate = g.Count(e => e.AttendanceStatus!.Code == "L"),
                DaysAbsent = g.Count(e => e.AttendanceStatus!.Code == "A"),
                DaysExcused = g.Count(e => e.AttendanceStatus!.Code == "E"),
            })
            .FirstOrDefaultAsync(ct);

        return new StudentAttendanceSummaryDto
        {
            StudentId = filter.StudentId,
            StudentName = student is null
                ? string.Empty
                : (student.FirstName + " " + student.LastName).Trim(),
            StudentAdmissionNumber = student?.AdmissionNumber ?? string.Empty,
            DaysCounted = counts?.DaysCounted ?? 0,
            DaysPresent = counts?.DaysPresent ?? 0,
            DaysLate = counts?.DaysLate ?? 0,
            DaysAbsent = counts?.DaysAbsent ?? 0,
            DaysExcused = counts?.DaysExcused ?? 0,
        };
    }

    public async Task<ClassAttendanceSummaryDto> GetClassSummaryAsync(Guid schoolClassId,
Guid? termId, CancellationToken ct = default)
    {
        var schoolClass = await db.SchoolClasses
            .Where(c => c.Id == schoolClassId)
            .Select(c => new { c.Name })
            .FirstOrDefaultAsync(ct);

        var enrolmentQuery = db.Enrolments
            .Where(e => e.SchoolClassId == schoolClassId);

        var students = await enrolmentQuery
            .Select(e => new
            {
                e.StudentId,
                StudentName = e.Student!.FirstName + " " + e.Student!.LastName,
                e.Student!.AdmissionNumber,
            })
            .Distinct()
            .ToListAsync(ct);

        var summaries = new List<StudentAttendanceSummaryDto>();
        foreach (var s in students)
        {
            var summary = await GetStudentSummaryAsync(new StudentAttendanceSummaryFilter
            {
                StudentId = s.StudentId,
                TermId = termId,
            }, ct);
```

```
            summary.StudentName = s.StudentName.Trim();
            summary.StudentAdmissionNumber = s.AdmissionNumber;
            summaries.Add(summary);
        }

        return new ClassAttendanceSummaryDto
        {
            SchoolClassId = schoolClassId,
            SchoolClassName = schoolClass?.Name ?? string.Empty,
            Students = summaries.OrderBy(s => s.StudentName).ToList(),
        };
    }

    private static IQueryable<DailyAttendanceEntryDto>
ProjectEntries(IQueryable<DailyAttendanceEntry> q) =>
        q.OrderBy(e => e.Student!.FirstName).ThenBy(e => e.Student!.LastName)
            .Select(e => new DailyAttendanceEntryDto
            {
                Id = e.Id,
                RegisterId = e.RegisterId,
                StudentId = e.StudentId,
                StudentName = (e.Student!.FirstName + " " + e.Student!.LastName).Trim(),
                StudentAdmissionNumber = e.Student!.AdmissionNumber,
                AttendanceStatusId = e.AttendanceStatusId,
                AttendanceStatusName = e.AttendanceStatus!.Name,
                AttendanceStatusCode = e.AttendanceStatus!.Code,
                CountsAsPresent = e.AttendanceStatus!.CountsAsPresent,
                ArrivalTime = e.ArrivalTime,
                Remarks = e.Remarks,
            });
}
```

Three patterns are worth calling out:

- OpenAsync resolves the term automatically. It first tries to find a Term whose StartDate ≤ Date ≤ EndDate; if no term covers the date (typical for a date that lands in a vacation), it falls back to the current term in the session. If neither resolves, it returns a friendly error asking the admin to set up a term first.
- OpenAsync is idempotent. Calling it twice for the same (class, date) returns the existing register's Id without duplicating rows. The unique index makes this the safest approach.
- BulkSetAsync diffs against existing rows. It loads the existing entries into a dictionary keyed by StudentId, then for each request entry it either updates the matching row or inserts a new one. The pattern is simple and correct under concurrent edits because the UI only ever submits the full set.

6.2 SubjectAttendanceService.cs

SubjectAttendanceService follows the same shape as the daily service, with the weekday-vs-date validation in OpenAsync that section 2.6 described. There is no summary query because per-subject summaries are deferred.

Listing — src/NaijaPrimeSchool.Infrastructure/Services/SubjectAttendanceService.cs

```
using Microsoft.EntityFrameworkCore;
using NaijaPrimeSchool.Application.Attendance;
using NaijaPrimeSchool.Application.Attendance.Dtos;
using NaijaPrimeSchool.Application.Common;
using NaijaPrimeSchool.Domain.Attendance;
using NaijaPrimeSchool.Infrastructure.Persistence;

namespace NaijaPrimeSchool.Infrastructure.Services;

public class SubjectAttendanceService(ApplicationDbContext db) : ISubjectAttendanceService
{
    private static IQueryable<SubjectAttendanceSessionDto>
ProjectSession(IQueryable<SubjectAttendanceSession> q) =>
        q.Select(s => new SubjectAttendanceSessionDto
        {
            Id = s.Id,
            TimetableEntryId = s.TimetableEntryId,
            SchoolClassId = s.TimetableEntry!.SchoolClassId,
            SchoolClassName = s.TimetableEntry!.SchoolClass!.Name,
            SubjectId = s.TimetableEntry!.SubjectId,
            SubjectName = s.TimetableEntry!.Subject!.Name,
            SubjectCode = s.TimetableEntry!.Subject!.Code,
            TermId = s.TimetableEntry!.TermId,
            TermName = s.TimetableEntry!.Term!.TermType!.Name,
            SessionId = s.TimetableEntry!.Term!.SessionId,
            SessionName = s.TimetableEntry!.Term!.Session!.Name,
            WeekDayId = s.TimetableEntry!.WeekDayId,
            WeekDayName = s.TimetableEntry!.WeekDay!.Name,
            TimetablePeriodId = s.TimetableEntry!.TimetablePeriodId,
            PeriodName = s.TimetableEntry!.TimetablePeriod!.Name,
            PeriodStart = s.TimetableEntry!.TimetablePeriod!.StartTime,
            PeriodEnd = s.TimetableEntry!.TimetablePeriod!.EndTime,
            Date = s.Date,
            TakenById = s.TakenById,
            TakenByName = s.TakenBy == null
                ? null
                : (s.TakenBy.FirstName + " " + s.TakenBy.LastName).Trim(),
            TakenOn = s.TakenOn,
            IsSubmitted = s.IsSubmitted,
            SubmittedOn = s.SubmittedOn,
            Notes = s.Notes,
            PresentCount = s.Entries.Count(e => e.AttendanceStatus!.CountsAsPresent),
            AbsentCount = s.Entries.Count(e => !e.AttendanceStatus!.CountsAsPresent),
            TotalCount = s.Entries.Count,
        });

    public async Task<IReadOnlyList<SubjectAttendanceSessionDto>>
ListAsync(SubjectSessionListFilter filter, CancellationToken ct = default)
    {
        var q = db.SubjectAttendanceSessions.AsQueryable();
        if (filter.TimetableEntryId.HasValue)
            q = q.Where(s => s.TimetableEntryId == filter.TimetableEntryId.Value);
        if (filter.SchoolClassId.HasValue)
            q = q.Where(s => s.TimetableEntry!.SchoolClassId ==
filter.SchoolClassId.Value);
```

```
        if (filter.TermId.HasValue)
            q = q.Where(s => s.TimetableEntry!.TermId == filter.TermId.Value);
        if (filter.SubjectId.HasValue)
            q = q.Where(s => s.TimetableEntry!.SubjectId == filter.SubjectId.Value);
        if (filter.FromDate.HasValue)
            q = q.Where(s => s.Date >= filter.FromDate.Value);
        if (filter.ToDate.HasValue)
            q = q.Where(s => s.Date <= filter.ToDate.Value);

        return await ProjectSession(q.OrderByDescending(s => s.Date)
                .ThenBy(s => s.TimetableEntry!.TimetablePeriod!.DisplayOrder))
            .ToListAsync(ct);
    }

    public async Task<SubjectAttendanceSessionDetailDto?> GetByIdAsync(Guid id,
CancellationToken ct = default)
    {
        var session = await ProjectSession(db.SubjectAttendanceSessions.Where(s => s.Id ==
id))
            .FirstOrDefaultAsync(ct);
        if (session is null) return null;

        var entries = await ProjectEntries(db.SubjectAttendanceEntries.Where(e =>
e.SessionId == id))
            .ToListAsync(ct);

        return new SubjectAttendanceSessionDetailDto { Session = session, Entries = entries
};
    }

    public async Task<SubjectAttendanceSessionDetailDto?> GetForEntryDateAsync(Guid
timetableEntryId, DateOnly date, CancellationToken ct = default)
    {
        var sessionId = await db.SubjectAttendanceSessions
            .Where(s => s.TimetableEntryId == timetableEntryId && s.Date == date)
            .Select(s => s.Id)
            .FirstOrDefaultAsync(ct);

        if (sessionId == Guid.Empty) return null;
        return await GetByIdAsync(sessionId, ct);
    }

    public async Task<OperationResult<Guid>> OpenAsync(OpenSubjectSessionRequest request,
CancellationToken ct = default)
    {
        var entry = await db.TimetableEntries
            .Include(e => e.SchoolClass)
            .Include(e => e.WeekDay)
            .Include(e => e.Term)
            .FirstOrDefaultAsync(e => e.Id == request.TimetableEntryId, ct);
        if (entry is null) return OperationResult<Guid>.Failure("Timetable entry not
found.");

        // Sanity: the date should fall inside the entry's term and on the entry's weekday.
```

```
        if (request.Date < entry.Term!.StartDate || request.Date > entry.Term!.EndDate)
            return OperationResult<Guid>.Failure(
                "Selected date is outside the term this lesson belongs to.");

        var weekDayName = entry.WeekDay!.Name;
        var actualDayOfWeek = request.Date.DayOfWeek.ToString();
        if (!string.Equals(weekDayName, actualDayOfWeek,
StringComparison.OrdinalIgnoreCase))
            return OperationResult<Guid>.Failure(
                $"Selected date is a {actualDayOfWeek} but this lesson runs on
{weekDayName}.");

        var existing = await db.SubjectAttendanceSessions
            .FirstOrDefaultAsync(s => s.TimetableEntryId == request.TimetableEntryId &&
s.Date == request.Date, ct);
        if (existing is not null)
            return OperationResult<Guid>.Success(existing.Id);

        var session = new SubjectAttendanceSession
        {
            TimetableEntryId = request.TimetableEntryId,
            Date = request.Date,
            TakenById = request.TakenById ?? entry.TeacherId,
            TakenOn = request.TakenById.HasValue || entry.TeacherId.HasValue ?
DateTimeOffset.UtcNow : null,
            IsSubmitted = false,
        };
        db.SubjectAttendanceSessions.Add(session);

        var defaultStatus = await db.AttendanceStatuses
            .OrderBy(s => s.DisplayOrder)
            .FirstOrDefaultAsync(ct);
        if (defaultStatus is null)
            return OperationResult<Guid>.Failure("Attendance statuses are not seeded.");

        var enrolledStudents = await db.Enrolments
            .Where(e => e.SchoolClassId == entry.SchoolClassId
                        && e.EnrolledOn <= request.Date
                        && (e.WithdrawnOn == null || e.WithdrawnOn >= request.Date))
            .Select(e => e.StudentId)
            .ToListAsync(ct);

        foreach (var studentId in enrolledStudents)
        {
            db.SubjectAttendanceEntries.Add(new SubjectAttendanceEntry
            {
                Session = session,
                StudentId = studentId,
                AttendanceStatusId = defaultStatus.Id,
            });
        }

        await db.SaveChangesAsync(ct);
        return OperationResult<Guid>.Success(session.Id);
```

```
    }

    public async Task<OperationResult> SetEntryAsync(UpsertSubjectEntryRequest request,
CancellationToken ct = default)
    {
        var session = await db.SubjectAttendanceSessions.FirstOrDefaultAsync(s => s.Id ==
request.SessionId, ct);
        if (session is null) return OperationResult.Failure("Session not found.");
        if (session.IsSubmitted) return OperationResult.Failure("Session is submitted.
Reopen it first.");

        if (!await db.AttendanceStatuses.AnyAsync(s => s.Id == request.AttendanceStatusId,
ct))
            return OperationResult.Failure("Attendance status not valid.");

        if (!await db.Students.AnyAsync(s => s.Id == request.StudentId, ct))
            return OperationResult.Failure("Student not found.");

        var entry = await db.SubjectAttendanceEntries
            .FirstOrDefaultAsync(e => e.SessionId == request.SessionId && e.StudentId ==
request.StudentId, ct);

        if (entry is null)
        {
            entry = new SubjectAttendanceEntry
            {
                SessionId = request.SessionId,
                StudentId = request.StudentId,
                AttendanceStatusId = request.AttendanceStatusId,
                Remarks = request.Remarks,
            };
            db.SubjectAttendanceEntries.Add(entry);
        }
        else
        {
            entry.AttendanceStatusId = request.AttendanceStatusId;
            entry.Remarks = request.Remarks;
        }

        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    public async Task<OperationResult> BulkSetAsync(BulkSetSubjectAttendanceRequest
request, CancellationToken ct = default)
    {
        var session = await db.SubjectAttendanceSessions.FirstOrDefaultAsync(s => s.Id ==
request.SessionId, ct);
        if (session is null) return OperationResult.Failure("Session not found.");
        if (session.IsSubmitted) return OperationResult.Failure("Session is submitted.
Reopen it first.");

        var validStatusIds = await db.AttendanceStatuses.Select(s => s.Id).ToListAsync(ct);
        var statusSet = validStatusIds.ToHashSet();
```

```
        var existing = await db.SubjectAttendanceEntries
            .Where(e => e.SessionId == request.SessionId)
            .ToDictionaryAsync(e => e.StudentId, ct);

        foreach (var item in request.Entries)
        {
            if (!statusSet.Contains(item.AttendanceStatusId))
                return OperationResult.Failure("One or more attendance statuses are
invalid.");

            if (existing.TryGetValue(item.StudentId, out var entry))
            {
                entry.AttendanceStatusId = item.AttendanceStatusId;
                entry.Remarks = item.Remarks;
            }
            else
            {
                db.SubjectAttendanceEntries.Add(new SubjectAttendanceEntry
                {
                    SessionId = request.SessionId,
                    StudentId = item.StudentId,
                    AttendanceStatusId = item.AttendanceStatusId,
                    Remarks = item.Remarks,
                });
            }
        }

        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    public async Task<OperationResult> SubmitAsync(Guid sessionId, CancellationToken ct =
default)
    {
        var session = await db.SubjectAttendanceSessions.FirstOrDefaultAsync(s => s.Id ==
sessionId, ct);
        if (session is null) return OperationResult.Failure("Session not found.");
        if (session.IsSubmitted) return OperationResult.Success();

        session.IsSubmitted = true;
        session.SubmittedOn = DateTimeOffset.UtcNow;
        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    public async Task<OperationResult> ReopenAsync(Guid sessionId, CancellationToken ct =
default)
    {
        var session = await db.SubjectAttendanceSessions.FirstOrDefaultAsync(s => s.Id ==
sessionId, ct);
        if (session is null) return OperationResult.Failure("Session not found.");

        session.IsSubmitted = false;
```

```
        session.SubmittedOn = null;
        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    public async Task<OperationResult> SoftDeleteAsync(Guid sessionId, CancellationToken ct
= default)
    {
        var session = await db.SubjectAttendanceSessions.FirstOrDefaultAsync(s => s.Id ==
sessionId, ct);
        if (session is null) return OperationResult.Failure("Session not found.");

        if (session.IsSubmitted)
            return OperationResult.Failure("Submitted sessions cannot be deleted. Reopen
first.");

        db.SubjectAttendanceSessions.Remove(session);
        await db.SaveChangesAsync(ct);
        return OperationResult.Success();
    }

    private static IQueryable<SubjectAttendanceEntryDto>
ProjectEntries(IQueryable<SubjectAttendanceEntry> q) =>
        q.OrderBy(e => e.Student!.FirstName).ThenBy(e => e.Student!.LastName)
            .Select(e => new SubjectAttendanceEntryDto
            {
                Id = e.Id,
                SessionId = e.SessionId,
                StudentId = e.StudentId,
                StudentName = (e.Student!.FirstName + " " + e.Student!.LastName).Trim(),
                StudentAdmissionNumber = e.Student!.AdmissionNumber,
                AttendanceStatusId = e.AttendanceStatusId,
                AttendanceStatusName = e.AttendanceStatus!.Name,
                AttendanceStatusCode = e.AttendanceStatus!.Code,
                CountsAsPresent = e.AttendanceStatus!.CountsAsPresent,
                Remarks = e.Remarks,
            });
}
```

6.3 LookupService — the new method

Excerpt — LookupService.GetAttendanceStatusesAsync

```
    public async Task<IReadOnlyList<LookupDto>>
GetAttendanceStatusesAsync(CancellationToken ct = default) =>
        await db.AttendanceStatuses
            .OrderBy(s => s.DisplayOrder)
            .Select(s => new LookupDto { Id = s.Id, Name = s.Name, Code = s.Code })
            .ToListAsync(ct);
```

6.4 DI registration

DependencyInjection.cs picks up two new lines.

Excerpt — DependencyInjection.cs (sprint 4 additions)

```
            services.AddScoped<IDailyAttendanceService, DailyAttendanceService>();
            services.AddScoped<ISubjectAttendanceService, SubjectAttendanceService>();

            return services;
```

7. The EF Core migration

A single migration named Attendance adds five new tables (AttendanceStatuses, DailyAttendanceRegisters, DailyAttendanceEntries, SubjectAttendanceSessions, SubjectAttendanceEntries) and the indexes that go with them. It was generated with:

```
dotnet ef migrations add Attendance \
  --project src/NaijaPrimeSchool.Infrastructure \
  --startup-project src/NaijaPrimeSchool.Web \
  --output-dir Persistence/Migrations
```

On a fresh checkout the migration runs at startup via DatabaseInitializer.MigrateAsync. The full Up() method is embedded below for reference.

Excerpt — Up() of 20260430212205_Attendance.cs

```
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "AttendanceStatuses",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    Name = table.Column<string>(type: "nvarchar(40)", maxLength: 40,
nullable: false),
                    Code = table.Column<string>(type: "nvarchar(5)", maxLength: 5,
nullable: false),
                    DisplayOrder = table.Column<int>(type: "int", nullable: false),
                    CountsAsPresent = table.Column<bool>(type: "bit", nullable: false),
                    CreatedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: false),
                    CreatedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true),
                    ModifiedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    ModifiedBy = table.Column<string>(type: "nvarchar(100)", maxLength:
100, nullable: true),
                    IsDeleted = table.Column<bool>(type: "bit", nullable: false),
                    DeletedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    DeletedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AttendanceStatuses", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "DailyAttendanceRegisters",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    SchoolClassId = table.Column<Guid>(type: "uniqueidentifier", nullable:
```

```
false),
                    TermId = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    Date = table.Column<DateOnly>(type: "date", nullable: false),
                    TakenById = table.Column<Guid>(type: "uniqueidentifier", nullable:
true),
                    TakenOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    IsSubmitted = table.Column<bool>(type: "bit", nullable: false),
                    SubmittedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    Notes = table.Column<string>(type: "nvarchar(500)", maxLength: 500,
nullable: true),
                    CreatedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: false),
                    CreatedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true),
                    ModifiedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    ModifiedBy = table.Column<string>(type: "nvarchar(100)", maxLength:
100, nullable: true),
                    IsDeleted = table.Column<bool>(type: "bit", nullable: false),
                    DeletedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    DeletedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_DailyAttendanceRegisters", x => x.Id);
                    table.ForeignKey(
                        name: "FK_DailyAttendanceRegisters_SchoolClasses_SchoolClassId",
                        column: x => x.SchoolClassId,
                        principalTable: "SchoolClasses",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                    table.ForeignKey(
                        name: "FK_DailyAttendanceRegisters_Terms_TermId",
                        column: x => x.TermId,
                        principalTable: "Terms",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                    table.ForeignKey(
                        name: "FK_DailyAttendanceRegisters_Users_TakenById",
                        column: x => x.TakenById,
                        principalTable: "Users",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.SetNull);
                });

            migrationBuilder.CreateTable(
                name: "SubjectAttendanceSessions",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    TimetableEntryId = table.Column<Guid>(type: "uniqueidentifier",
```

```
nullable: false),
                    Date = table.Column<DateOnly>(type: "date", nullable: false),
                    TakenById = table.Column<Guid>(type: "uniqueidentifier", nullable:
true),
                    TakenOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    IsSubmitted = table.Column<bool>(type: "bit", nullable: false),
                    SubmittedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    Notes = table.Column<string>(type: "nvarchar(500)", maxLength: 500,
nullable: true),
                    CreatedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: false),
                    CreatedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true),
                    ModifiedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    ModifiedBy = table.Column<string>(type: "nvarchar(100)", maxLength:
100, nullable: true),
                    IsDeleted = table.Column<bool>(type: "bit", nullable: false),
                    DeletedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    DeletedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_SubjectAttendanceSessions", x => x.Id);
                    table.ForeignKey(
                        name:
"FK_SubjectAttendanceSessions_TimetableEntries_TimetableEntryId",
                        column: x => x.TimetableEntryId,
                        principalTable: "TimetableEntries",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                    table.ForeignKey(
                        name: "FK_SubjectAttendanceSessions_Users_TakenById",
                        column: x => x.TakenById,
                        principalTable: "Users",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.SetNull);
                });

            migrationBuilder.CreateTable(
                name: "DailyAttendanceEntries",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    RegisterId = table.Column<Guid>(type: "uniqueidentifier", nullable:
false),
                    StudentId = table.Column<Guid>(type: "uniqueidentifier", nullable:
false),
                    AttendanceStatusId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
                    ArrivalTime = table.Column<TimeOnly>(type: "time", nullable: true),
```

```
                    Remarks = table.Column<string>(type: "nvarchar(300)", maxLength: 300,
nullable: true),
                    CreatedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: false),
                    CreatedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true),
                    ModifiedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    ModifiedBy = table.Column<string>(type: "nvarchar(100)", maxLength:
100, nullable: true),
                    IsDeleted = table.Column<bool>(type: "bit", nullable: false),
                    DeletedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    DeletedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_DailyAttendanceEntries", x => x.Id);
                    table.ForeignKey(
                        name:
"FK_DailyAttendanceEntries_AttendanceStatuses_AttendanceStatusId",
                        column: x => x.AttendanceStatusId,
                        principalTable: "AttendanceStatuses",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                    table.ForeignKey(
                        name:
"FK_DailyAttendanceEntries_DailyAttendanceRegisters_RegisterId",
                        column: x => x.RegisterId,
                        principalTable: "DailyAttendanceRegisters",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_DailyAttendanceEntries_Students_StudentId",
                        column: x => x.StudentId,
                        principalTable: "Students",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                });

            migrationBuilder.CreateTable(
                name: "SubjectAttendanceEntries",
                columns: table => new
                {
                    Id = table.Column<Guid>(type: "uniqueidentifier", nullable: false),
                    SessionId = table.Column<Guid>(type: "uniqueidentifier", nullable:
false),
                    StudentId = table.Column<Guid>(type: "uniqueidentifier", nullable:
false),
                    AttendanceStatusId = table.Column<Guid>(type: "uniqueidentifier",
nullable: false),
                    Remarks = table.Column<string>(type: "nvarchar(300)", maxLength: 300,
nullable: true),
                    CreatedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
```

```
nullable: false),
                    CreatedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true),
                    ModifiedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    ModifiedBy = table.Column<string>(type: "nvarchar(100)", maxLength:
100, nullable: true),
                    IsDeleted = table.Column<bool>(type: "bit", nullable: false),
                    DeletedOn = table.Column<DateTimeOffset>(type: "datetimeoffset",
nullable: true),
                    DeletedBy = table.Column<string>(type: "nvarchar(100)", maxLength: 100,
nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_SubjectAttendanceEntries", x => x.Id);
                    table.ForeignKey(
                        name:
"FK_SubjectAttendanceEntries_AttendanceStatuses_AttendanceStatusId",
                        column: x => x.AttendanceStatusId,
                        principalTable: "AttendanceStatuses",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                    table.ForeignKey(
                        name: "FK_SubjectAttendanceEntries_Students_StudentId",
                        column: x => x.StudentId,
                        principalTable: "Students",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Restrict);
                    table.ForeignKey(
                        name:
"FK_SubjectAttendanceEntries_SubjectAttendanceSessions_SessionId",
                        column: x => x.SessionId,
                        principalTable: "SubjectAttendanceSessions",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateIndex(
                name: "IX_AttendanceStatuses_Code",
                table: "AttendanceStatuses",
                column: "Code",
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_AttendanceStatuses_Name",
                table: "AttendanceStatuses",
                column: "Name",
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceEntries_AttendanceStatusId",
                table: "DailyAttendanceEntries",
                column: "AttendanceStatusId");
```

```
            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceEntries_IsDeleted",
                table: "DailyAttendanceEntries",
                column: "IsDeleted");

            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceEntries_RegisterId_StudentId",
                table: "DailyAttendanceEntries",
                columns: new[] { "RegisterId", "StudentId" },
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceEntries_StudentId",
                table: "DailyAttendanceEntries",
                column: "StudentId");

            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceRegisters_Date",
                table: "DailyAttendanceRegisters",
                column: "Date");

            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceRegisters_IsDeleted",
                table: "DailyAttendanceRegisters",
                column: "IsDeleted");

            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceRegisters_SchoolClassId_Date",
                table: "DailyAttendanceRegisters",
                columns: new[] { "SchoolClassId", "Date" },
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceRegisters_TakenById",
                table: "DailyAttendanceRegisters",
                column: "TakenById");

            migrationBuilder.CreateIndex(
                name: "IX_DailyAttendanceRegisters_TermId",
                table: "DailyAttendanceRegisters",
                column: "TermId");

            migrationBuilder.CreateIndex(
                name: "IX_SubjectAttendanceEntries_AttendanceStatusId",
                table: "SubjectAttendanceEntries",
                column: "AttendanceStatusId");

            migrationBuilder.CreateIndex(
                name: "IX_SubjectAttendanceEntries_IsDeleted",
                table: "SubjectAttendanceEntries",
                column: "IsDeleted");

            migrationBuilder.CreateIndex(
                name: "IX_SubjectAttendanceEntries_SessionId_StudentId",
```

```
                table: "SubjectAttendanceEntries",
                columns: new[] { "SessionId", "StudentId" },
                unique: true);

            migrationBuilder.CreateIndex(
                name: "IX_SubjectAttendanceEntries_StudentId",
                table: "SubjectAttendanceEntries",
                column: "StudentId");

            migrationBuilder.CreateIndex(
                name: "IX_SubjectAttendanceSessions_Date",
                table: "SubjectAttendanceSessions",
                column: "Date");

            migrationBuilder.CreateIndex(
                name: "IX_SubjectAttendanceSessions_IsDeleted",
                table: "SubjectAttendanceSessions",
                column: "IsDeleted");

            migrationBuilder.CreateIndex(
                name: "IX_SubjectAttendanceSessions_TakenById",
                table: "SubjectAttendanceSessions",
                column: "TakenById");

            migrationBuilder.CreateIndex(
                name: "IX_SubjectAttendanceSessions_TimetableEntryId_Date",
                table: "SubjectAttendanceSessions",
                columns: new[] { "TimetableEntryId", "Date" },
                unique: true);
        }

        /// <inheritdoc />
        protected override void Down(MigrationBuilder migrationBuilder)
```

Notice the sequence: AttendanceStatuses first, then the two parent tables (DailyAttendanceRegisters and SubjectAttendanceSessions), then the two entry tables. EF Core computes the ordering automatically from the foreign-key graph; we did not have to specify it.

7.1 Indexes created

- Unique on AttendanceStatuses.Name and AttendanceStatuses.Code (2 unique).
- Unique on DailyAttendanceRegisters.(SchoolClassId, Date) (1 unique).
- Unique on DailyAttendanceEntries.(RegisterId, StudentId) (1 unique).
- Unique on SubjectAttendanceSessions.(TimetableEntryId, Date) (1 unique).
- Unique on SubjectAttendanceEntries.(SessionId, StudentId) (1 unique).
- Plus IsDeleted indexes on every soft-deletable table (5).
- Plus a Date index on each of the two parent tables for date-range queries (2).

8. Seeding the AttendanceStatus lookup

DatabaseInitializer picks up a new SeedAttendanceLookupsAsync method that is invoked between SeedFamilyLookupsAsync and SeedRolesAsync. It seeds six rows: Present, Late, Excused, Sick, Absent, Suspended. CountsAsPresent is true for Present and Late, false for the rest.

Excerpt — SeedAttendanceLookupsAsync

```
    private static async Task SeedAttendanceLookupsAsync(ApplicationDbContext db,
CancellationToken ct)
    {
        if (!await db.AttendanceStatuses.IgnoreQueryFilters().AnyAsync(ct))
        {
            (string Name, string Code, bool CountsAsPresent)[] statuses =
            [
                ("Present",   "P",  true),
                ("Late",      "L",  true),
                ("Excused",   "E",  false),
                ("Sick",      "S",  false),
                ("Absent",    "A",  false),
                ("Suspended", "SP", false),
            ];
            for (var i = 0; i < statuses.Length; i++)
            {
                db.AttendanceStatuses.Add(new AttendanceStatus
                {
                    Name = statuses[i].Name,
                    Code = statuses[i].Code,
                    DisplayOrder = i + 1,
                    CountsAsPresent = statuses[i].CountsAsPresent,
                });
            }
        }

        await db.SaveChangesAsync(ct);
    }

    private static async Task SeedFamilyLookupsAsync(ApplicationDbContext db,
CancellationToken ct)
    {
        if (!await db.Relationships.IgnoreQueryFilters().AnyAsync(ct))
        {
            string[] relationships =
            [
                "Father",
                "Mother",
                "Stepfather",
                "Stepmother",
                "Grandfather",
                "Grandmother",
                "Uncle",
                "Aunt",
                "Guardian",
                "Other",
            ];
```

```
            for (var i = 0; i < relationships.Length; i++)
            {
                db.Relationships.Add(new Relationship { Name = relationships[i],
DisplayOrder = i + 1 });
            }
        }

        if (!await db.EnrolmentStatuses.IgnoreQueryFilters().AnyAsync(ct))
        {
            string[] statuses =
            [
                "Active",
                "Suspended",
                "Withdrawn",
                "Transferred",
                "Graduated",
            ];
            for (var i = 0; i < statuses.Length; i++)
            {
                db.EnrolmentStatuses.Add(new EnrolmentStatus { Name = statuses[i],
DisplayOrder = i + 1 });
            }
        }

        if (!await db.BloodGroups.IgnoreQueryFilters().AnyAsync(ct))
        {
            string[] groups = ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-",
"Unknown"];
            for (var i = 0; i < groups.Length; i++)
            {
                db.BloodGroups.Add(new BloodGroup { Name = groups[i], DisplayOrder = i +
1 });
            }
        }

        if (!await db.MaritalStatuses.IgnoreQueryFilters().AnyAsync(ct))
        {
            string[] statuses = ["Single", "Married", "Divorced", "Widowed", "Separated"];
            for (var i = 0; i < statuses.Length; i++)
            {
                db.MaritalStatuses.Add(new MaritalStatus { Name = statuses[i], DisplayOrder
= i + 1 });
            }
        }

        await db.SaveChangesAsync(ct);
    }

    private static async Task SeedAcademicLookupsAsync(ApplicationDbContext db,
CancellationToken ct)
    {
        if (!await db.TermTypes.IgnoreQueryFilters().AnyAsync(ct))
        {
            db.TermTypes.AddRange(
```

```
                new TermType { Name = "First Term", DisplayOrder = 1 },
                new TermType { Name = "Second Term", DisplayOrder = 2 },
                new TermType { Name = "Third Term", DisplayOrder = 3 });
        }

        if (!await db.ClassLevels.IgnoreQueryFilters().AnyAsync(ct))
        {
            string[] levels =
            [
                "Creche",
                "Pre-Nursery",
                "Nursery 1",
                "Nursery 2",
                "KG 1",
                "KG 2",
                "Primary 1",
                "Primary 2",
                "Primary 3",
                "Primary 4",
                "Primary 5",
                "Primary 6",
            ];
            for (var i = 0; i < levels.Length; i++)
            {
                db.ClassLevels.Add(new ClassLevel { Name = levels[i], DisplayOrder = i +
1 });
            }
        }

        if (!await db.WeekDays.IgnoreQueryFilters().AnyAsync(ct))
        {
            db.WeekDays.AddRange(
                new WeekDay { Name = "Monday",    ShortName = "Mon", DisplayOrder = 1 },
                new WeekDay { Name = "Tuesday",   ShortName = "Tue", DisplayOrder = 2 },
                new WeekDay { Name = "Wednesday", ShortName = "Wed", DisplayOrder = 3 },
                new WeekDay { Name = "Thursday",  ShortName = "Thu", DisplayOrder = 4 },
                new WeekDay { Name = "Friday",    ShortName = "Fri", DisplayOrder = 5 });
        }

        if (!await db.TimetablePeriods.IgnoreQueryFilters().AnyAsync(ct))
        {
            (string Name, int Hour, int Minute, int DurationMinutes, int Order, bool
IsBreak)[] periods =
            [
                ("Period 1",      8, 0,  40, 1, false),
                ("Period 2",      8, 40, 40, 2, false),
                ("Period 3",      9, 20, 40, 3, false),
                ("Short Break",  10, 0,  20, 4, true),
                ("Period 4",     10, 20, 40, 5, false),
                ("Period 5",     11, 0,  40, 6, false),
                ("Lunch",        11, 40, 40, 7, true),
                ("Period 6",     12, 20, 40, 8, false),
                ("Period 7",     13, 0,  40, 9, false),
            ];
            foreach (var p in periods)
```

```
            {
                var start = new TimeOnly(p.Hour, p.Minute);
                db.TimetablePeriods.Add(new TimetablePeriod
                {
                    Name = p.Name,
                    StartTime = start,
                    EndTime = start.AddMinutes(p.DurationMinutes),
                    DisplayOrder = p.Order,
                    IsBreak = p.IsBreak,
                });
            }
        }

        await db.SaveChangesAsync(ct);
    }

    private static async Task SeedLookupsAsync(ApplicationDbContext db, CancellationToken
ct)
```

The seeder uses the same .IgnoreQueryFilters().AnyAsync() guard as the previous sprints' seeders — it only inserts if the table is empty (counting soft-deleted rows), so running the app a second time after a soft delete does not resurrect the missing row.

9. The Razor pages

Three new pages land in src/NaijaPrimeSchool.Web/Components/Pages/Attendance/. Two are register-taking pages (one daily, one per-subject), and the third is a class-level summary.

9.1 Page roster

```
src/NaijaPrimeSchool.Web/Components/Pages/Attendance/
├── DailyAttendance.razor      ← /attendance/daily   (open + mark + submit)
├── SubjectAttendance.razor    ← /attendance/subject (per-lesson register)
└── AttendanceSummary.razor    ← /attendance/summary (class % view)
```

All three pages are gated to SuperAdmin + HeadTeacher + Teacher. Teachers genuinely need write access to the daily and subject pages — they are the people taking attendance. Bursars, Storekeepers, Parents, and Students do not see the panel.

9.2 DailyAttendance.razor — the workhorse

DailyAttendance.razor is the page teachers will use every morning. The top filter bar narrows down the (session, class, date) tuple; the 'Open register' button creates (or opens an existing) register for that combination.

Listing — src/NaijaPrimeSchool.Web/Components/Pages/Attendance/DailyAttendance.razor

```
@page "/attendance/daily"
@attribute [Authorize(Roles = $"{Roles.SuperAdmin},{Roles.HeadTeacher},{Roles.Teacher}")]
@rendermode InteractiveServer

@inject IDailyAttendanceService AttendanceService
@inject ISessionService SessionService
@inject ILookupService LookupService
@inject NotificationService Notification
@inject DialogService Dialog

<PageTitle>Daily attendance · Naija Prime School</PageTitle>

<div class="nps-page-header">
    <div>
        <h1>Daily attendance</h1>
        <p>Pick a class and a date, then mark each pupil's attendance status. Submit the
register when you're done.</p>
    </div>
</div>

<RadzenCard class="nps-card" Style="margin-bottom: 1rem;">
    <div class="nps-filter-bar">
        <div class="nps-field">
            <label>Session</label>
            <RadzenDropDown TValue="Guid?" Data="@sessions" TextProperty="Name"
ValueProperty="Id"
                            @bind-Value="sessionId" Placeholder="Pick a session"
                            Change="@(_ => SessionChangedAsync())" Style="min-width:
200px;" />
        </div>
```

```
        <div class="nps-field">
            <label>Class</label>
            <RadzenDropDown TValue="Guid?" Data="@classes" TextProperty="Name"
ValueProperty="Id"
                            @bind-Value="schoolClassId" Placeholder="Pick a class"
                            Change="@(_ => LoadAsync())" Style="min-width: 240px;" />
        </div>
        <div class="nps-field">
            <label>Date</label>
            <RadzenDatePicker TValue="DateTime?" @bind-Value="date" DateFormat="d MMM yyyy"
ShowTime="false"
                              Change="@(_ => LoadAsync())" Style="min-width: 180px;" />
        </div>
        <RadzenButton Text="Open register" Icon="how_to_reg" Variant="Variant.Flat"
                      ButtonStyle="ButtonStyle.Primary" Disabled="@(schoolClassId is null
|| date is null)"
                      Click="@OpenAsync" />
    </div>
</RadzenCard>

@if (loading)
{
    <p>Loading register...</p>
}
else if (detail is null)
{
    <RadzenCard class="nps-card">
        <p style="color: var(--nps-ink-500);">
            No register found for the selected class and date. Click <strong>Open
register</strong>
            to create one — pupils currently enrolled in the class will be pre-loaded as
Present.
        </p>
    </RadzenCard>
}
else
{
    <RadzenCard class="nps-card">
        <div style="display:flex; justify-content: space-between; align-items: flex-start;
margin-bottom: 1rem; gap:1rem; flex-wrap:wrap;">
            <div>
                <h3 style="margin:0;">@detail.Register.SchoolClassName ·
@detail.Register.Date.ToString("dddd, d MMMM yyyy")</h3>
                <p style="margin:.25rem 0 0; color: var(--nps-ink-500);">
                    Term: <strong>@detail.Register.TermName</strong> · Session:
<strong>@detail.Register.SessionName</strong>
                </p>
                <div style="margin-top:.5rem; display:flex; gap:.5rem; flex-wrap:wrap;">
                    <RadzenBadge Text="@($"Present: {detail.Register.PresentCount}")"
BadgeStyle="BadgeStyle.Success" Variant="Variant.Flat" />
                    <RadzenBadge Text="@($"Absent: {detail.Register.AbsentCount}")"
BadgeStyle="BadgeStyle.Danger" Variant="Variant.Flat" />
                    <RadzenBadge Text="@($"Late: {detail.Register.LateCount}")"
BadgeStyle="BadgeStyle.Warning" Variant="Variant.Flat" />
                    <RadzenBadge Text="@($"Total: {detail.Register.TotalCount}")"
```

```
BadgeStyle="BadgeStyle.Info" Variant="Variant.Flat" />
                </div>
            </div>
            <div class="nps-inline-actions">
                @if (detail.Register.IsSubmitted)
                {
                    <RadzenBadge Text="@($"Submitted {detail.Register.SubmittedOn:d MMM
HH:mm}")"
                                 BadgeStyle="BadgeStyle.Success" Variant="Variant.Flat" />
                    <RadzenButton Text="Reopen" Icon="lock_open"
ButtonStyle="ButtonStyle.Warning"
                                  Variant="Variant.Flat" Click="@ReopenAsync" />
                }
                else
                {
                    <RadzenButton Text="Save changes" Icon="save"
ButtonStyle="ButtonStyle.Light"
                                  Variant="Variant.Flat" Click="@SaveAsync"
Disabled="@saving" />
                    <RadzenButton Text="Submit register" Icon="check_circle"
ButtonStyle="ButtonStyle.Primary"
                                  Click="@SubmitAsync" Disabled="@saving" />
                }
                <RadzenButton Icon="delete" ButtonStyle="ButtonStyle.Danger"
Variant="Variant.Flat"
                              Click="@DeleteAsync" title="Delete register" />
            </div>
        </div>

        <RadzenDataGrid TItem="DailyAttendanceEntryDto" Data="@detail.Entries"
AllowPaging="false"
                        EmptyText="No pupils on this register.">
            <Columns>
                <RadzenDataGridColumn TItem="DailyAttendanceEntryDto" Title="Pupil">
                    <Template Context="e">
                        <strong>@e.StudentName</strong>
                        <br />
                        <small style="color: var(--nps-
ink-500);">@e.StudentAdmissionNumber</small>
                    </Template>
                </RadzenDataGridColumn>
                <RadzenDataGridColumn TItem="DailyAttendanceEntryDto" Title="Status"
Width="200px">
                    <Template Context="e">
                        <RadzenDropDown TValue="Guid" Data="@statuses" TextProperty="Name"
ValueProperty="Id"
                                        Value="@e.AttendanceStatusId"
                                        Change="@(args => OnStatusChanged(e, (Guid)args!))"
                                        Disabled="@detail.Register.IsSubmitted"
                                        Style="width: 100%;" />
                    </Template>
                </RadzenDataGridColumn>
                <RadzenDataGridColumn TItem="DailyAttendanceEntryDto" Title="Arrival"
Width="140px">
                    <Template Context="e">
```

```
                        @if (e.AttendanceStatusCode == "L")
                        {
                            <RadzenTextBox Value="@(e.ArrivalTime?.ToString("HH:mm"))"
                                           Change="@(v => OnArrivalChanged(e, v))"
                                           Disabled="@detail.Register.IsSubmitted"
                                           Placeholder="HH:mm" Style="width: 100%;" />
                        }
                        else
                        {
                            <span style="color: var(--nps-ink-500);">—</span>
                        }
                    </Template>
                </RadzenDataGridColumn>
                <RadzenDataGridColumn TItem="DailyAttendanceEntryDto" Title="Remarks">
                    <Template Context="e">
                        <RadzenTextBox @bind-Value="e.Remarks"
                                       Disabled="@detail.Register.IsSubmitted"
                                       Style="width: 100%;" />
                    </Template>
                </RadzenDataGridColumn>
            </Columns>
        </RadzenDataGrid>
    </RadzenCard>
}

@code {
    private IReadOnlyList<SessionDto> sessions = [];
    private IReadOnlyList<LookupDto> classes = [];
    private IReadOnlyList<LookupDto> statuses = [];

    private Guid? sessionId;
    private Guid? schoolClassId;
    private DateTime? date = DateTime.Today;

    private DailyAttendanceRegisterDetailDto? detail;
    private bool loading;
    private bool saving;

    protected override async Task OnInitializedAsync()
    {
        sessions = await SessionService.ListAsync();
        sessionId = sessions.FirstOrDefault(s => s.IsCurrent)?.Id ??
sessions.FirstOrDefault()?.Id;
        classes = await LookupService.GetClassesForSessionAsync(sessionId);
        statuses = await LookupService.GetAttendanceStatusesAsync();
    }

    private async Task SessionChangedAsync()
    {
        schoolClassId = null;
        detail = null;
        classes = await LookupService.GetClassesForSessionAsync(sessionId);
    }
```

```
    private async Task LoadAsync()
    {
        if (schoolClassId is null || date is null)
        {
            detail = null;
            return;
        }

        loading = true;
        try
        {
            detail = await AttendanceService.GetForClassDateAsync(schoolClassId.Value,
DateOnly.FromDateTime(date.Value));
        }
        finally
        {
            loading = false;
            StateHasChanged();
        }
    }

    private async Task OpenAsync()
    {
        if (schoolClassId is null || date is null) return;

        saving = true;
        try
        {
            var result = await AttendanceService.OpenAsync(new OpenDailyRegisterRequest
            {
                SchoolClassId = schoolClassId.Value,
                Date = DateOnly.FromDateTime(date.Value),
            });

            if (result.Succeeded)
            {
                Notification.Notify(NotificationSeverity.Success, "Opened", "Register
opened.");
                await LoadAsync();
            }
            else
            {
                Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
            }
        }
        finally
        {
            saving = false;
        }
    }

    private void OnStatusChanged(DailyAttendanceEntryDto entry, Guid statusId)
    {
```

```
        entry.AttendanceStatusId = statusId;
        var status = statuses.FirstOrDefault(s => s.Id == statusId);
        if (status is not null)
        {
            entry.AttendanceStatusCode = status.Code ?? string.Empty;
            entry.AttendanceStatusName = status.Name;
        }
    }

    private void OnArrivalChanged(DailyAttendanceEntryDto entry, string? value)
    {
        if (string.IsNullOrWhiteSpace(value))
        {
            entry.ArrivalTime = null;
            return;
        }
        if (TimeOnly.TryParse(value, out var parsed))
            entry.ArrivalTime = parsed;
    }

    private async Task SaveAsync()
    {
        if (detail is null) return;

        saving = true;
        try
        {
            var result = await AttendanceService.BulkSetAsync(new
BulkSetDailyAttendanceRequest
            {
                RegisterId = detail.Register.Id,
                Entries = detail.Entries.Select(e => new UpsertDailyEntryRequest
                {
                    Id = e.Id,
                    RegisterId = e.RegisterId,
                    StudentId = e.StudentId,
                    AttendanceStatusId = e.AttendanceStatusId,
                    ArrivalTime = e.ArrivalTime,
                    Remarks = e.Remarks,
                }).ToList(),
            });

            if (result.Succeeded)
            {
                Notification.Notify(NotificationSeverity.Success, "Saved", "Attendance
saved.");
                await LoadAsync();
            }
            else
            {
                Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
            }
        }
```

```
        finally
        {
            saving = false;
        }
    }

    private async Task SubmitAsync()
    {
        if (detail is null) return;

        await SaveAsync();

        var result = await AttendanceService.SubmitAsync(detail.Register.Id);
        if (result.Succeeded)
        {
            Notification.Notify(NotificationSeverity.Success, "Submitted", "Register
submitted.");
            await LoadAsync();
        }
        else
        {
            Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
        }
    }

    private async Task ReopenAsync()
    {
        if (detail is null) return;
        var result = await AttendanceService.ReopenAsync(detail.Register.Id);
        if (result.Succeeded)
        {
            Notification.Notify(NotificationSeverity.Success, "Reopened", "Register
reopened for editing.");
            await LoadAsync();
        }
        else
        {
            Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
        }
    }

    private async Task DeleteAsync()
    {
        if (detail is null) return;
        var confirm = await Dialog.Confirm(
            $"Delete the register for {detail.Register.SchoolClassName} on
{detail.Register.Date:d MMM yyyy}?",
            "Delete register",
            new ConfirmOptions { OkButtonText = "Delete", CancelButtonText = "Cancel" });
        if (confirm != true) return;

        var result = await AttendanceService.SoftDeleteAsync(detail.Register.Id);
```

```
        if (result.Succeeded)
        {
            Notification.Notify(NotificationSeverity.Success, "Deleted", "Register
removed.");
            detail = null;
        }
        else
        {
            Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
        }
    }
}
```

Three details deserve a moment:

- Status is rendered as a RadzenDropDown bound directly to the entry's AttendanceStatusId. The OnStatusChanged handler also updates the entry's Code locally so that the arrival-time column can re-render based on the new code without a server round-trip.
- Arrival time appears only when the status code is 'L'. The TimeOnly value is parsed from a free-text 'HH:mm' field; bad input simply leaves the value alone, which is friendly enough for a primary-school workflow.
- Submit calls Save first, then SubmitAsync. This makes the two-button workflow simple: edit in the grid, click Submit, and the page does the right thing without asking the teacher to remember to click Save.

9.3 SubjectAttendance.razor — the per-lesson page

SubjectAttendance.razor has a two-stage flow. First the teacher picks (term, class, date); the page lists every lesson on the timetable for that class on that weekday. Clicking 'Take attendance' on a lesson opens (or creates) a SubjectAttendanceSession for that (timetable entry, date) pair and renders the register.

Listing — src/NaijaPrimeSchool.Web/Components/Pages/Attendance/SubjectAttendance.razor

```
@page "/attendance/subject"
@attribute [Authorize(Roles = $"{Roles.SuperAdmin},{Roles.HeadTeacher},{Roles.Teacher}")]
@rendermode InteractiveServer

@inject ISubjectAttendanceService AttendanceService
@inject ITimetableService Timetable
@inject ITermService TermService
@inject ISchoolClassService ClassService
@inject ISessionService SessionService
@inject ILookupService LookupService
@inject NotificationService Notification
@inject DialogService Dialog

<PageTitle>Subject attendance · Naija Prime School</PageTitle>

<div class="nps-page-header">
    <div>
        <h1>Subject attendance</h1>
```

```
        <p>Pick a term, class and date — every lesson scheduled for that day on the
timetable will appear. Click any lesson to take attendance for it.</p>
    </div>
</div>

<RadzenCard class="nps-card" Style="margin-bottom: 1rem;">
    <div class="nps-filter-bar">
        <div class="nps-field">
            <label>Term</label>
            <RadzenDropDown TValue="Guid?" Data="@terms" TextProperty="Display"
ValueProperty="Id"
                            @bind-Value="termId" Placeholder="Pick a term"
                            Change="@(_ => TermChangedAsync())" Style="min-width:
240px;" />
        </div>
        <div class="nps-field">
            <label>Class</label>
            <RadzenDropDown TValue="Guid?" Data="@classes" TextProperty="Name"
ValueProperty="Id"
                            @bind-Value="schoolClassId" Placeholder="Pick a class"
                            Change="@(_ => LoadLessonsAsync())" Style="min-width:
240px;" />
        </div>
        <div class="nps-field">
            <label>Date</label>
            <RadzenDatePicker TValue="DateTime?" @bind-Value="date" DateFormat="d MMM yyyy"
ShowTime="false"
                              Change="@(_ => LoadLessonsAsync())" Style="min-width: 180px;"
/>
        </div>
    </div>
</RadzenCard>

@if (lessons.Count == 0 && termId is not null && schoolClassId is not null && date is not
null)
{
    <RadzenCard class="nps-card">
        <p style="color: var(--nps-ink-500);">
            No lessons scheduled for this class on @date?.ToString("dddd, d MMMM yyyy").
        </p>
    </RadzenCard>
}
else if (lessons.Count > 0)
{
    <RadzenCard class="nps-card">
        <h3 style="margin: 0 0 1rem;">Lessons on @date?.ToString("dddd, d MMMM yyyy")</h3>
        <RadzenDataGrid TItem="TimetableEntryDto" Data="@lessons" AllowPaging="false"
                        EmptyText="No lessons.">
            <Columns>
                <RadzenDataGridColumn TItem="TimetableEntryDto" Title="Period"
Width="200px">
                    <Template Context="t">
                        <strong>@t.PeriodName</strong>
                        <br />
                        <small style="color: var(--nps-
```

```
ink-500);">@t.PeriodStart.ToString("HH:mm") – @t.PeriodEnd.ToString("HH:mm")</small>
                    </Template>
                </RadzenDataGridColumn>
                <RadzenDataGridColumn TItem="TimetableEntryDto" Title="Subject">
                    <Template Context="t">
                        <strong>@t.SubjectName</strong>
                        <br />
                        <small style="color: var(--nps-ink-500);">@t.SubjectCode
@(string.IsNullOrEmpty(t.TeacherName) ? "" : "· " + t.TeacherName)</small>
                    </Template>
                </RadzenDataGridColumn>
                <RadzenDataGridColumn TItem="TimetableEntryDto" Title="Action"
Width="220px" TextAlign="TextAlign.Right" Sortable="false">
                    <Template Context="t">
                        <RadzenButton Text="Take attendance" Icon="how_to_reg"
                                      ButtonStyle="ButtonStyle.Primary"
Size="ButtonSize.Small"
                                      Click="@(() => OpenSessionAsync(t))" />
                    </Template>
                </RadzenDataGridColumn>
            </Columns>
        </RadzenDataGrid>
    </RadzenCard>
}

@if (detail is not null)
{
    <RadzenCard class="nps-card" Style="margin-top: 1rem;">
        <div style="display:flex; justify-content: space-between; align-items: flex-start;
margin-bottom: 1rem; gap:1rem; flex-wrap:wrap;">
            <div>
                <h3 style="margin:0;">@detail.Session.SubjectName ·
@detail.Session.PeriodName</h3>
                <p style="margin:.25rem 0 0; color: var(--nps-ink-500);">
                    @detail.Session.SchoolClassName · @detail.Session.Date.ToString("dddd,
d MMMM yyyy") ·
                    @detail.Session.PeriodStart.ToString("HH:mm")–
@detail.Session.PeriodEnd.ToString("HH:mm")
                </p>
                <div style="margin-top:.5rem; display:flex; gap:.5rem; flex-wrap:wrap;">
                    <RadzenBadge Text="@($"Present: {detail.Session.PresentCount}")"
BadgeStyle="BadgeStyle.Success" Variant="Variant.Flat" />
                    <RadzenBadge Text="@($"Absent: {detail.Session.AbsentCount}")"
BadgeStyle="BadgeStyle.Danger" Variant="Variant.Flat" />
                    <RadzenBadge Text="@($"Total: {detail.Session.TotalCount}")"
BadgeStyle="BadgeStyle.Info" Variant="Variant.Flat" />
                </div>
            </div>
            <div class="nps-inline-actions">
                @if (detail.Session.IsSubmitted)
                {
                    <RadzenBadge Text="@($"Submitted {detail.Session.SubmittedOn:d MMM
HH:mm}")"
                                 BadgeStyle="BadgeStyle.Success" Variant="Variant.Flat" />
                    <RadzenButton Text="Reopen" Icon="lock_open"
```

```
ButtonStyle="ButtonStyle.Warning"
                                Variant="Variant.Flat" Click="@ReopenAsync" />
                }
                else
                {
                    <RadzenButton Text="Save changes" Icon="save"
ButtonStyle="ButtonStyle.Light"
                                Variant="Variant.Flat" Click="@SaveAsync"
Disabled="@saving" />
                    <RadzenButton Text="Submit" Icon="check_circle"
ButtonStyle="ButtonStyle.Primary"
                                Click="@SubmitAsync" Disabled="@saving" />
                }
                <RadzenButton Text="Close" Icon="close" Variant="Variant.Flat"
                              ButtonStyle="ButtonStyle.Light" Click="@(() => detail =
null)" />
            </div>
        </div>

        <RadzenDataGrid TItem="SubjectAttendanceEntryDto" Data="@detail.Entries"
AllowPaging="false"
                        EmptyText="No pupils on this register.">
            <Columns>
                <RadzenDataGridColumn TItem="SubjectAttendanceEntryDto" Title="Pupil">
                    <Template Context="e">
                        <strong>@e.StudentName</strong>
                        <br />
                        <small style="color: var(--nps-
ink-500);">@e.StudentAdmissionNumber</small>
                    </Template>
                </RadzenDataGridColumn>
                <RadzenDataGridColumn TItem="SubjectAttendanceEntryDto" Title="Status"
Width="200px">
                    <Template Context="e">
                        <RadzenDropDown TValue="Guid" Data="@statuses" TextProperty="Name"
ValueProperty="Id"
                                        Value="@e.AttendanceStatusId"
                                        Change="@(args => OnStatusChanged(e, (Guid)args!))"
                                        Disabled="@detail.Session.IsSubmitted"
                                        Style="width: 100%;" />
                    </Template>
                </RadzenDataGridColumn>
                <RadzenDataGridColumn TItem="SubjectAttendanceEntryDto" Title="Remarks">
                    <Template Context="e">
                        <RadzenTextBox @bind-Value="e.Remarks"
                                       Disabled="@detail.Session.IsSubmitted"
                                       Style="width: 100%;" />
                    </Template>
                </RadzenDataGridColumn>
            </Columns>
        </RadzenDataGrid>
    </RadzenCard>
}

@code {
```

```
    private record TermOption(Guid Id, string Display);

    private List<TermOption> terms = [];
    private IReadOnlyList<LookupDto> classes = [];
    private IReadOnlyList<LookupDto> statuses = [];
    private List<TimetableEntryDto> lessons = [];

    private Guid? termId;
    private Guid? schoolClassId;
    private DateTime? date = DateTime.Today;

    private SubjectAttendanceSessionDetailDto? detail;
    private bool saving;

    protected override async Task OnInitializedAsync()
    {
        var allTerms = await TermService.ListAsync();
        terms = allTerms
            .OrderByDescending(t => t.IsCurrent)
            .ThenByDescending(t => t.StartDate)
            .Select(t => new TermOption(t.Id, $"{t.SessionName} · {t.TermTypeName}"))
            .ToList();

        var current = allTerms.FirstOrDefault(t => t.IsCurrent);
        termId = current?.Id ?? terms.FirstOrDefault()?.Id;

        if (current is not null)
            classes = await LookupService.GetClassesForSessionAsync(current.SessionId);
        else
            classes = await LookupService.GetClassesForSessionAsync();

        statuses = await LookupService.GetAttendanceStatusesAsync();
    }

    private async Task TermChangedAsync()
    {
        var allTerms = await TermService.ListAsync();
        var t = allTerms.FirstOrDefault(x => x.Id == termId);
        if (t is null) return;
        classes = await LookupService.GetClassesForSessionAsync(t.SessionId);
        schoolClassId = null;
        lessons = [];
        detail = null;
    }

    private async Task LoadLessonsAsync()
    {
        detail = null;
        if (termId is null || schoolClassId is null || date is null)
        {
            lessons = [];
            return;
        }
```

```
        var entries = await Timetable.ListEntriesAsync(new TimetableQuery
        {
            TermId = termId.Value,
            SchoolClassId = schoolClassId.Value,
        });

        var dayName = date.Value.DayOfWeek.ToString();
        lessons = entries
            .Where(e => string.Equals(e.WeekDayName, dayName,
StringComparison.OrdinalIgnoreCase))
            .OrderBy(e => e.PeriodOrder)
            .ToList();
    }

    private async Task OpenSessionAsync(TimetableEntryDto t)
    {
        if (date is null) return;
        var result = await AttendanceService.OpenAsync(new OpenSubjectSessionRequest
        {
            TimetableEntryId = t.Id,
            Date = DateOnly.FromDateTime(date.Value),
        });

        if (result.Succeeded)
        {
            detail = await AttendanceService.GetByIdAsync(result.Data);
        }
        else
        {
            Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
        }
    }

    private void OnStatusChanged(SubjectAttendanceEntryDto entry, Guid statusId)
    {
        entry.AttendanceStatusId = statusId;
        var status = statuses.FirstOrDefault(s => s.Id == statusId);
        if (status is not null)
        {
            entry.AttendanceStatusCode = status.Code ?? string.Empty;
            entry.AttendanceStatusName = status.Name;
        }
    }

    private async Task SaveAsync()
    {
        if (detail is null) return;
        saving = true;
        try
        {
            var result = await AttendanceService.BulkSetAsync(new
BulkSetSubjectAttendanceRequest
            {
```

```
                SessionId = detail.Session.Id,
                Entries = detail.Entries.Select(e => new UpsertSubjectEntryRequest
                {
                    Id = e.Id,
                    SessionId = e.SessionId,
                    StudentId = e.StudentId,
                    AttendanceStatusId = e.AttendanceStatusId,
                    Remarks = e.Remarks,
                }).ToList(),
            });

            if (result.Succeeded)
            {
                Notification.Notify(NotificationSeverity.Success, "Saved", "Attendance
saved.");
                detail = await AttendanceService.GetByIdAsync(detail.Session.Id);
            }
            else
            {
                Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
            }
        }
        finally
        {
            saving = false;
        }
    }

    private async Task SubmitAsync()
    {
        if (detail is null) return;
        await SaveAsync();
        var result = await AttendanceService.SubmitAsync(detail.Session.Id);
        if (result.Succeeded)
        {
            Notification.Notify(NotificationSeverity.Success, "Submitted", "Session
submitted.");
            detail = await AttendanceService.GetByIdAsync(detail.Session.Id);
        }
        else
        {
            Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
        }
    }

    private async Task ReopenAsync()
    {
        if (detail is null) return;
        var result = await AttendanceService.ReopenAsync(detail.Session.Id);
        if (result.Succeeded)
        {
            Notification.Notify(NotificationSeverity.Success, "Reopened", "Session
reopened.");
```

```
            detail = await AttendanceService.GetByIdAsync(detail.Session.Id);
        }
        else
        {
            Notification.Notify(NotificationSeverity.Error, "Failed", string.Join("; ",
result.Errors));
        }
    }
}
```

The page leans on sprint 2's TimetableService to find lessons. There is a deliberate filter on WeekDayName matching DateTime.DayOfWeek.ToString() — Saturday lessons would only appear if a Saturday WeekDay row existed and the date being marked is a Saturday.

9.4 AttendanceSummary.razor — the percentage view

AttendanceSummary.razor renders the per-class summary the IDailyAttendanceService.GetClassSummaryAsync method returns. Each pupil's percentage is bucketed into green (≥ 90%), amber (≥ 75%), or red (< 75%) — a common rule of thumb in Nigerian primary schools.

Listing — src/NaijaPrimeSchool.Web/Components/Pages/Attendance/AttendanceSummary.razor

```
@page "/attendance/summary"
@attribute [Authorize(Roles = $"{Roles.SuperAdmin},{Roles.HeadTeacher},{Roles.Teacher}")]
@rendermode InteractiveServer

@inject IDailyAttendanceService DailyAttendance
@inject ISessionService SessionService
@inject ITermService TermService
@inject ILookupService LookupService

<PageTitle>Attendance summary · Naija Prime School</PageTitle>

<div class="nps-page-header">
    <div>
        <h1>Attendance summary</h1>
        <p>Daily-attendance percentages per pupil for the selected class and term.</p>
    </div>
</div>

<RadzenCard class="nps-card" Style="margin-bottom: 1rem;">
    <div class="nps-filter-bar">
        <div class="nps-field">
            <label>Session</label>
            <RadzenDropDown TValue="Guid?" Data="@sessions" TextProperty="Name"
ValueProperty="Id"
                            @bind-Value="sessionId" Placeholder="Pick a session"
                            Change="@(_ => SessionChangedAsync())" Style="min-width:
200px;" />
        </div>
        <div class="nps-field">
            <label>Term</label>
            <RadzenDropDown TValue="Guid?" Data="@terms" TextProperty="TermTypeName"
ValueProperty="Id"
```

```
                            @bind-Value="termId" AllowClear="true" Placeholder="All terms
in session"
                            Change="@(_ => LoadAsync())" Style="min-width: 200px;" />
        </div>
        <div class="nps-field">
            <label>Class</label>
            <RadzenDropDown TValue="Guid?" Data="@classes" TextProperty="Name"
ValueProperty="Id"
                            @bind-Value="schoolClassId" Placeholder="Pick a class"
                            Change="@(_ => LoadAsync())" Style="min-width: 240px;" />
        </div>
    </div>
</RadzenCard>

@if (summary is not null)
{
    <RadzenCard class="nps-card">
        <h3 style="margin: 0 0 1rem;">@summary.SchoolClassName</h3>
        <RadzenDataGrid TItem="StudentAttendanceSummaryDto" Data="@summary.Students"
AllowPaging="true" PageSize="25"
                        EmptyText="No attendance data yet for this selection.">
            <Columns>
                <RadzenDataGridColumn TItem="StudentAttendanceSummaryDto" Title="Pupil">
                    <Template Context="s">
                        <strong>@s.StudentName</strong>
                        <br />
                        <small style="color: var(--nps-
ink-500);">@s.StudentAdmissionNumber</small>
                    </Template>
                </RadzenDataGridColumn>
                <RadzenDataGridColumn TItem="StudentAttendanceSummaryDto" Title="Days
counted" Property="DaysCounted" Width="140px" />
                <RadzenDataGridColumn TItem="StudentAttendanceSummaryDto" Title="Present"
Property="DaysPresent" Width="100px" />
                <RadzenDataGridColumn TItem="StudentAttendanceSummaryDto" Title="Late"
Property="DaysLate" Width="100px" />
                <RadzenDataGridColumn TItem="StudentAttendanceSummaryDto" Title="Absent"
Property="DaysAbsent" Width="100px" />
                <RadzenDataGridColumn TItem="StudentAttendanceSummaryDto" Title="Excused"
Property="DaysExcused" Width="100px" />
                <RadzenDataGridColumn TItem="StudentAttendanceSummaryDto" Title="Present
rate" Width="160px">
                    <Template Context="s">
                        <RadzenBadge Text="@($"{s.PresentRate}%")"
                                     BadgeStyle="@(s.PresentRate >= 90 ? BadgeStyle.Success
                                                   : s.PresentRate >= 75 ?
BadgeStyle.Warning
                                                   : BadgeStyle.Danger)"
                                     Variant="Variant.Flat" />
                    </Template>
                </RadzenDataGridColumn>
            </Columns>
        </RadzenDataGrid>
    </RadzenCard>
}
```

```
@code {
    private IReadOnlyList<SessionDto> sessions = [];
    private IReadOnlyList<TermDto> terms = [];
    private IReadOnlyList<LookupDto> classes = [];

    private Guid? sessionId;
    private Guid? termId;
    private Guid? schoolClassId;

    private ClassAttendanceSummaryDto? summary;

    protected override async Task OnInitializedAsync()
    {
        sessions = await SessionService.ListAsync();
        sessionId = sessions.FirstOrDefault(s => s.IsCurrent)?.Id ??
sessions.FirstOrDefault()?.Id;
        await SessionChangedAsync();
    }

    private async Task SessionChangedAsync()
    {
        if (sessionId is null) return;
        terms = await TermService.ListAsync(sessionId);
        termId = terms.FirstOrDefault(t => t.IsCurrent)?.Id;
        classes = await LookupService.GetClassesForSessionAsync(sessionId);
        schoolClassId = null;
        summary = null;
    }

    private async Task LoadAsync()
    {
        if (schoolClassId is null) { summary = null; return; }
        summary = await DailyAttendance.GetClassSummaryAsync(schoolClassId.Value, termId);
    }
}
```

10. Navigation, imports, and authorization

10.1 NavMenu — a new Attendance panel

NavMenu.razor picks up a third role-gated panel between Family and the Finance/Inventory placeholders. Three entries live inside it: Daily attendance, Subject attendance, Summary. The whole panel is wrapped in an AuthorizeView that requires SuperAdmin, HeadTeacher, or Teacher.

Excerpt — NavMenu.razor

```
            <RadzenPanelMenuItem Text="Attendance" Icon="event_available" Expanded="true">
                <RadzenPanelMenuItem Text="Daily attendance" Icon="today"
Path="/attendance/daily" />
                <RadzenPanelMenuItem Text="Subject attendance" Icon="schedule"
Path="/attendance/subject" />
                <RadzenPanelMenuItem Text="Summary" Icon="analytics"
Path="/attendance/summary" />
            </RadzenPanelMenuItem>
        </Authorized>
    </AuthorizeView>
```

10.2 _Imports.razor

Excerpt — _Imports.razor

```
@using NaijaPrimeSchool.Application.Attendance
@using NaijaPrimeSchool.Application.Attendance.Dtos
```

10.3 The page-class / inject-field collision

There is one Blazor-specific gotcha worth flagging. The Razor page's class name is derived from the file name, so DailyAttendance.razor compiles into a class called DailyAttendance. If you @inject IDailyAttendanceService with the field name DailyAttendance, the C# compiler rejects it with CS0542 because a member cannot share a name with its enclosing type.

Both attendance pages therefore inject their service with the field name AttendanceService. It is a small thing, but it is the kind of error that takes ten minutes to recognise the first time.

11. Lifecycle of a daily register

Walking a single register from creation to submission is the clearest way to see how all the layers cooperate. Imagine the Primary 1A class teacher taking morning attendance on Monday 4 May 2026.

11.1 Teacher opens /attendance/daily

- Authorization fires: the user must be SuperAdmin, HeadTeacher, or Teacher. If they are not, ASP.NET Core returns 403.
- OnInitializedAsync calls SessionService.ListAsync, LookupService.GetClassesForSessionAsync, and GetAttendanceStatusesAsync — three small queries, all on tiny tables.
- The session dropdown defaults to the current session, and the date picker defaults to today.

11.2 Teacher picks Primary 1A and clicks Open register

- DailyAttendanceService.OpenAsync looks up the class to verify it exists and to read its SessionId.
- It looks for a Term where StartDate ≤ 2026-05-04 ≤ EndDate in that session. Suppose it finds Second Term.
- It checks for an existing register on (class, 2026-05-04). None exists, so it creates one with TermId pointing at Second Term.
- It loads the lowest-DisplayOrder AttendanceStatus, which is 'Present'.
- It loads every Enrolment in the class with EnrolledOn ≤ 2026-05-04 AND (WithdrawnOn IS NULL OR WithdrawnOn ≥ 2026-05-04). Suppose it returns 28 pupils.
- It creates 28 DailyAttendanceEntry rows, each pointing at the new register and the Present status.
- SaveChangesAsync commits register + 28 entries in one transaction. The audit trail captures the teacher's username on every CreatedBy column.

11.3 Teacher marks two pupils Late and one Absent

- Each dropdown change fires OnStatusChanged in the Razor file. The change is local to the page model — no service call yet.
- For the two Late pupils, the arrival-time text box appears next to the dropdown. The teacher types 08:15 and 08:22 respectively.
- Teacher clicks 'Save changes'. BulkSetAsync sends all 28 entries to the server.
- The service loads the existing 28 entries into a dictionary keyed by StudentId. For each request entry, it finds the matching existing row and updates the AttendanceStatusId, ArrivalTime, and Remarks. The two Late rows and the one Absent row are modified; the other 25 rows show no change. EF Core's change tracker emits an UPDATE statement only for the dirty rows.
- ApplicationDbContext.SaveChanges stamps ModifiedOn/By on each modified row.

11.4 Teacher clicks Submit

- SubmitAsync (after a final SaveAsync that catches any edits made between the last Save and the Submit click) sets IsSubmitted = true and SubmittedOn = now on the register row.
- The grid re-renders. Every dropdown is now disabled. The 'Save' and 'Submit' buttons are replaced with a 'Submitted just now' badge and a 'Reopen' button.

11.5 Half an hour later — a parent calls

- An admin (SuperAdmin or HeadTeacher) clicks Reopen. ReopenAsync flips IsSubmitted = false and clears SubmittedOn.
- The grid re-renders with editable dropdowns. The admin changes one pupil from Absent to Late, captures the arrival time, and clicks Submit again.
- ModifiedOn/By on the affected row records the second edit. The audit trail makes the post-submission edit visible.

12. Smoke-test walkthrough

Once the build is green and the migration has applied, here is the end-to-end smoke test you can run against a fresh checkout. It covers the happy path and a couple of important error paths.

12.1 Build, migrate, run

```
dotnet restore
dotnet build NaijaPrimeSchool.slnx
dotnet run --project src/NaijaPrimeSchool.Web
```

The first run applies migrations and seeds the AttendanceStatus lookup. Sign in as superadmin@naijaprimeschool.ng / Admin@12345.

12.2 Set up the prerequisites

7. Academics → Sessions: create a session for the current academic year if one does not exist; mark it current.
8. Academics → Terms: create at least one term covering today's date.
9. Academics → Classes: create Primary 1A.
10. Academics → Subjects: create Mathematics.
11. Academics → Periods: keep the seeded periods or tweak them.
12. Academics → Timetable: pick the term and Primary 1A; assign Mathematics to the Monday Period 1 cell.
13. Family → Add Student: create at least one pupil and enrol them in Primary 1A.

12.3 Take a daily register

14. Attendance → Daily attendance.
15. Pick the current session, Primary 1A, and today's date.
16. Click Open register. The pre-loaded grid shows the pupil(s) you enrolled, all set to Present.
17. Change one pupil to Late and capture an arrival time.
18. Click Save changes — confirmation toast appears.
19. Click Submit register. The grid disables; the badge flips to 'Submitted just now'.
20. Click Reopen. The grid re-enables. Click Submit again to lock it back.

12.4 Take a subject register

21. Attendance → Subject attendance.
22. Pick the term, Primary 1A, and a Monday date that covers the timetable entry you set up.
23. Click Take attendance on the Mathematics row.
24. Mark the pupil(s) and Submit.

12.5 Verify error paths

25. From Subject attendance, pick a Tuesday. The Mathematics row should not appear (it runs on Monday only).
26. Manually call OpenAsync via the daily page on a Saturday in a session that has no term covering Saturdays. The register cannot be opened — the service returns 'No term covers this date and no current term is set'.

27. Submit a register, then try to delete it from the page. The delete is refused with 'Submitted registers cannot be deleted. Reopen first.'

12.6 Verify the summary view

Attendance → Summary. Pick the session, term, and Primary 1A. Each pupil's row shows days-counted, days-present, days-absent, days-late, days-excused, and a green/amber/red percentage badge. If a pupil was absent on the day you marked, their percentage drops accordingly.

12.7 Verify soft-delete and audit

Connect to SQL Server and run:

```
SELECT Id, SchoolClassId, [Date], IsSubmitted,
       SubmittedOn, IsDeleted, DeletedOn
FROM DailyAttendanceRegisters;

SELECT Id, RegisterId, StudentId,
       AttendanceStatusId, ArrivalTime,
       CreatedOn, ModifiedOn
FROM DailyAttendanceEntries;
```

If you reopened a submitted register and changed a row, ModifiedOn on that DailyAttendanceEntry row is stamped. The register's SubmittedOn is null after the reopen and re-stamped after the second submit.

13. Troubleshooting and gotchas

13.1 'No term covers this date and no current term is set'

OpenAsync on the daily service refuses to create a register if the date has no term and the session has no current term. Set up a term first via /terms (sprint 2) or mark an existing term current.

13.2 'Selected date is a Wednesday but this lesson runs on Monday'

OpenAsync on the subject service refuses a date whose DayOfWeek does not match the timetable entry's WeekDay. Check the calendar; the timetable entry is correct.

13.3 'Submitted registers cannot be deleted'

Deleting a submitted register is intentionally a two-step: reopen it first, then delete. This forces the action to leave a clear audit trail.

13.4 The pupil list on a register is stale

The register pre-loads pupils at OpenAsync time. If you enrol a new pupil after opening the register, they will not appear automatically — that is by design, because the register is a snapshot of who the teacher saw that day. SetEntryAsync can still be called for the new pupil if you want to add them, but the page does not surface that flow today; deleting the register and reopening it is the simplest path.

13.5 'member names cannot be the same as their enclosing type' (CS0542)

If you copy the existing pages and rename the file without renaming the @inject field, the C# compiler will reject the build. The Razor page class name is derived from the file name; an inject field cannot share that name. Both attendance pages therefore use AttendanceService as the inject field name.

13.6 The summary page shows 0 days counted

GetClassSummaryAsync reads from DailyAttendanceEntry, not from SubjectAttendanceEntry. If you have only taken subject attendance for the class, the summary will show zeros until at least one daily register is submitted.

14. Forward-compatibility, today

Sprint 4 has deliberately left a few breadcrumbs that make the next sprints' work cleaner.

- AttendanceStatus.CountsAsPresent is the load-bearing flag for percentage calculations. Sprint 5 (results) will likely lean on attendance percentages as a feeder into the term report card; the column is ready.
- DailyAttendanceRegister.TermId means 'all attendance for first term' is a single index seek. Reporting queries do not need to span term boundaries by joining on date ranges.
- SubjectAttendanceSession.TimetableEntryId means 'all attendance for Mathematics in second term' is a join through TimetableEntry — fast and friendly.
- Reopening + the audit trail mean a parent portal (later sprint) can show 'attendance updated this morning at 08:42' without losing the original register's submission timestamp.
- TakenById is nullable, so SchoolClass.ClassTeacher is not load-bearing here. If a class has no class teacher assigned, attendance still works — TakenBy just stays null.

14.1 What might need a small refactor later

- GetClassSummaryAsync issues one query per pupil. Once classes have 30+ pupils and the school has 2000+ pupils total, this will need to become a single GROUP BY query in SQL. The lift is small (LINQ GroupBy on StudentId) but I deliberately kept the implementation simple here.
- The arrival-time field is a free-text 'HH:mm' input rather than a Radzen TimePicker. Radzen's TimePicker expects DateTime?, and converting through the binding boundary is fiddly enough to defer until the design system has a TimeOnly story.
- Per-subject summary reporting ("Adaeze missed three Maths lessons this term") is straightforward off the existing schema but the UI is missing. A /attendance/subject-summary page is on the backlog.
- Bulk editor for a whole week. Currently a teacher edits one date at a time. A 'mark this pupil absent for the whole week' workflow would be a useful add-on.

15. Appendix — files added or changed in sprint 4

Domain layer (new)	—
src/NaijaPrimeSchool.Domain/Attendance/AttendanceStatus.cs	Lookup.
src/NaijaPrimeSchool.Domain/Attendance/DailyAttendanceRegister.cs	Class × date root.
src/NaijaPrimeSchool.Domain/Attendance/DailyAttendanceEntry.cs	Register × pupil.
src/NaijaPrimeSchool.Domain/Attendance/SubjectAttendanceSession.cs	Timetable entry × date root.
src/NaijaPrimeSchool.Domain/Attendance/SubjectAttendanceEntry.cs	Session × pupil.
Domain layer (modified)	—
src/NaijaPrimeSchool.Domain/Academics/SchoolClass.cs	Added DailyAttendanceRegisters navigation.
src/NaijaPrimeSchool.Domain/Academics/Term.cs	Added DailyAttendanceRegisters navigation.
src/NaijaPrimeSchool.Domain/Academics/TimetableEntry.cs	Added AttendanceSessions navigation.
src/NaijaPrimeSchool.Domain/Family/Student.cs	Added DailyAttendanceEntries + SubjectAttendanceEntries navigations.
Application layer (new)	—
src/NaijaPrimeSchool.Application/Attendance/Dtos/DailyAttendanceDtos.cs	Daily register DTOs.
src/NaijaPrimeSchool.Application/Attendance/Dtos/SubjectAttendanceDtos.cs	Subject session DTOs.
src/NaijaPrimeSchool.Application/Attendance/Dtos/AttendanceSummaryDtos.cs	Per-pupil/class summary DTOs.
src/NaijaPrimeSchool.Application/Attendance/IDailyAttendanceService.cs	Daily register service contract.
src/NaijaPrimeSchool.Application/Attendance/ISubjectAttendanceService.cs	Subject session service contract.
Application layer (modified)	—
src/NaijaPrimeSchool.Application/Users/ILookupService.cs	Added GetAttendanceStatusesAsync.
Infrastructure layer (new)	—
src/NaijaPrimeSchool.Infrastructure/Services/DailyAttendanceService.cs	Daily register CRUD + summaries.
src/NaijaPrimeSchool.Infrastructure/Services/SubjectAttendanceService.cs	Subject session CRUD.
src/NaijaPrimeSchool.Infrastructure/Persistence/Migrations/20260430212205_Attendance.cs	EF migration adding 5 tables and 13 indexes.
Infrastructure layer (modified)	—
src/NaijaPrimeSchool.Infrastructure/DependencyInjection.cs	Registered the 2 new services.
src/NaijaPrimeSchool.Infrastructure/	Added 5 DbSets, ConfigureAttendance.

Persistence/ApplicationDbContext.cs	
src/NaijaPrimeSchool.Infrastructure/ Persistence/DatabaseInitializer.cs	Seeded AttendanceStatus.
src/NaijaPrimeSchool.Infrastructure/Services/ LookupService.cs	Added GetAttendanceStatusesAsync.
Web layer (new)	—
src/NaijaPrimeSchool.Web/Components/ Pages/Attendance/DailyAttendance.razor	Daily register editor.
src/NaijaPrimeSchool.Web/Components/ Pages/Attendance/SubjectAttendance.razor	Per-lesson register editor.
src/NaijaPrimeSchool.Web/Components/ Pages/Attendance/AttendanceSummary.razor	Class % view.
Web layer (modified)	—
src/NaijaPrimeSchool.Web/Components/ _Imports.razor	Added Attendance + Attendance.Dtos usings.
src/NaijaPrimeSchool.Web/Components/ Layout/NavMenu.razor	Added the Attendance panel.
Tooling (new)	—
tools/generate_sprint4_guide.py	This document's generator.

— End of the Sprint 4 implementation guide. The next sprint lands assessments and report cards on top of the data primitives established here.