1. Introduction

This report presents the design and implementation of a complete XML-based information system for a cycling tour operator. The objective of the project was to model, structure, and manage all the operational data required to organize guided cycling trips—ranging from clients, guides, destinations and cycling paths to tour packages, stages, bike rentals, maintenance logs, and bookings. Because tourism management involves highly interrelated entities and frequent cross-referencing between resources, XML was used as the core technology to express a rich, well-structured and interoperable data platform.
Beyond the conceptual modeling, the project required the construction of a fully modular XML Schema (XSD) defining precise data types, enumerations, constraints, and inter-entity relationships. A representative XML database compliant with this schema was produced to simulate the daily operations of the tour operator. The system also supports user-oriented functionalities such as searching for tour offers, browsing cycling paths, managing trip groups, checking bike availability, tracking maintenance, and reviewing booking status.
To exploit the database, six visualization scenarios were implemented using XSLT to generate HTML pages for different business needs, such as displaying activities per stage, grouping clients by experience level, listing available bikes, or reorganizing tours and stages. Two additional structural XSL transformations reorganize the XML tree into new XML formats, while two JSON-oriented XSLTs extract subsets of the database into JSON structures suitable for downstream processing.
Finally, the project integrates Python programs capable of loading XML files, validating them against the XSD, applying XSLT stylesheets, and reproducing one of the scenarios using the DOM API. Together, these components form a complete XML/XSD/XSLT/JSON/Python pipeline that demonstrates the relevance of XML technologies for real-world tourism and logistics management.

2. Workload Distribution

Each team member contributed equally to the project, as shown in the table. Tasks were divided logically based on the strengths of each member.

Team Member	**Tasks**
Kibet Vivian	Data Modeling & Report Writing
Zapfack Messiane Jasen Steve	XML Schema Architecture & Database
Adounkpe Abed Exauce Donatin	XSLT Visualization Scenarios
Aka N Guessan Antoine Paul David	XSLT Structure-Modification scenarios
Yobo Marcel Dylan Arnaud	Python Pipeline

3. Working Environment and Tools

We primarily worked with **Visual Studio Code (VS Code)** for editing XML, XSD, XSLT, JSON, and Python files, and **GitHub** for version control and collaborative development.

For validating and exploring schemas, we used notepad ++ plugins together with VSCode XML plugins. The Python components were implemented and executed using **Python 3.12** and supporting libraries such as lxml and xml.dom. Minidom.

Also xsl transformer test tool https://xslttest.appspot.com/

4. Modeling Principles and Choices

Our XML model follows a modular, hierarchical, and strongly typed structure designed to reflect the real complexity of a cycling tour operator while remaining maintainable and extensible. The platform is divided into five functional domains—clients & guides, tour packages, destinations & paths, bikes & maintenance, and bookings—each implemented in its own XSD module and imported into the root schema (main.xsd). This modular approach provides clear separation of concerns and allows each subsystem to evolve independently.

4.1 Modeling Principles

Modularity:

Each domain (e.g., clients, tours, bikes) is isolated in its own schema file. This allows reuse, independent validation, and easier debugging.

Strong typing and constraints:

We used numerous custom simple types (e.g., ExperienceLevel, DifficultyLevel, PriceType, LatitudeType) to enforce data correctness directly at the schema level.

Referential integrity with ID/IDREF:

Links between entities (e.g., TourPackage → Stage, Stage → Activity, Booking → Client) are enforced using ID and IDREF(S), preventing broken references and ensuring database consistency.

Real-world alignment:

Entities and relationships reflect how a real cycling tour operator functions:

- a TourPackage is composed of multiple Stages,
- each Stage uses CyclingPaths and offers Activities,
- Bikes include availability and maintenance history,
- Bookings link clients, trip groups, bikes, and tours.

We first designed the model in a **relational/UML style** to clearly represent the entities, attributes, and cardinalities. Then we adapted it to an **XML modeling approach**, using:

- xs:key / xs:keyref to enforce referential integrity
- xs:ID / xs:IDREF / xs:IDREFS for cross-references
- xs:simpleType restrictions for enums
- nested XML structures for compositions (e.g., TourPackage → Stages)
- For the JSON version, ID references are represented using arrays.

4.2 Advantages of our Modeling Choices

- It easy to **query all stages of a tour** or retrieve related entities.
- The definition of **TripGroup** is simple: each group is led by one guide and represents clients who booked a tour package.
- The use of enumerations enforces consistent values for difficulty, bike type, etc.
- The modularization into multiple XSD files improves readability and maintainability.
- Tours, stages, bikes, activities, and destinations can be added easily without modifying the structure.

4.3 Disadvantages / Trade-offs

- Some information may be duplicated (e.g., TourPackage contains stage IDs, but Stage also contains a tour ID). This may lead to inconsistencies if not validated properly.
- Number of participants in trip group could directly be quary with attribute current participants.

4.4 Modeling Problem Addressed and Our Solution

Problem Chosen: *How to model the relationship between Stages and CyclingPaths while ensuring correct directionality (from → to destinations).*

Challenges:

A stage uses one or several cycling paths, and each cycling path must always explicitly connect a start and end destination.

The model needed to enforce:

1. A path always has a defined `fromDestinationId` and `toDestinationId`.
2. A stage references the correct list of paths.
3. No stage should link destinations that contradict the path network.

Our Solution:

We modeled CyclingPath as an independent entity with two mandatory attributes: `fromDestinationId` and `toDestinationId`. Stages then reference paths using cyclingPathIds:

```
<cyclingPathIds>

    <xs:attribute name="fromDestinationId" type="xs:IDREF" use="required"/>

    <xs:attribute name="toDestinationId" type="xs:IDREF" use="required"/>

</cyclingPathIds>
```

This approach ensures:

- **Clear directional flow** of each route.
- **Full flexibility** to reuse paths across different tours.
- **A realistic representation** of a cycling network rather than embedding path information inside the stage.
- **Lightweight stages**, since they only store IDs instead of repeating path data.

This modeling choice made it easier to implement XSLT visualizations (e.g., displaying stage routes) and keep Python-based processing simple.

5. Scenario Implementation

This project implements **six XSLT visualization scenarios**, **two structure-modification scenarios**, and **two XML→JSON transformations** based on the XML dataset. The six visualization scenarios generate HTML outputs for: **Activities per Stage**, **Clients by Experience Level**, **Available Bikes**, **Destinations by Type**, **Bookings by Status**, and **Tours with their Stages**. These XSLT transformations demonstrate effective use of XPath querying, iteration, grouping, and HTML formatting to present meaningful insights from the dataset.

The two structure-modification scenarios reorganize XML data into alternative hierarchical layouts, showcasing the flexibility of XSLT for restructuring and recombining elements. Additionally, two JSON-oriented XSLT transformations extract selected information into JSON-like structures, which are further processed by Python scripts to generate summary reports. Together, these scenarios highlight how the XML model can support real-world operations such as managing tours, bookings, clients, destinations, and bike rentals.

6. Complex Scenario

Among the implemented transformations, the *Activities per Stage* scenario represents one of the most complex XSLT visualizations due to its multi-level traversal of the XML hierarchy and its need to group nested information into a readable HTML structure. This scenario extracts each stage of every tour package and displays the activities associated with that specific day, providing users with a clear overview of the daily program.

6.1 Complexity of the Scenario

The complexity arises from the hierarchical nature of the XML model.The XSLT must navigate these nested elements, apply multiple templates, and structure the information so that each stage is presented with its corresponding list of activities. Unlike scenarios that rely on simple grouping or attribute filtering, this transformation requires coordinated traversal of multiple levels of XML depth.

6.2 Transformation Logic

The transformation is structured around a sequence of templates, each responsible for processing a specific element of the data hierarchy. The root template initiates the process by selecting every stage in the XML file and preparing the overall structure of the output. For each stage, a dedicated template extracts key information such as the day number, title, and the list of activities associated with that stage.

Once a stage is selected, the transformation proceeds by applying templates to each of its activities. This ensures that activities are handled independently and rendered consistently. The activity template formats each activity into a readable form, displaying its title and description as part of an HTML list grouped under the correct stage.

To achieve this, the transformation relies on standard XSLT mechanisms such as template matching, element selection through XPath expressions, and template chaining. Each template focuses solely on one level of the hierarchy, allowing the transformation to remain modular, readable, and easy to maintain. The result is a clean, structured HTML output where activities are clearly grouped under their corresponding stages.

7. AI Assistance

ChatGPT was used as a supportive tool during the project. It helped clarify XML and XSLT concepts, assisted with debugging XSD syntax, and provided example structures to better understand modeling choices. It was also used to improve the grammar, structure, and clarity of the written report.

However, **all technical implementations—including the XML Schema, XML database, XSLT transformations, JSON outputs, and Python programs—were produced, tested, and validated manually by the team members**. ChatGPT did not generate any final code included in the submission.

Its role was limited to explanations, documentation assistance, and general guidance.

8. Conclusion

The project successfully demonstrates a full XML-based data processing pipeline, including modeling, validation, transformation, and Python integration. The architecture is modular, extensible, and suitable for real-world tourism data management.