**XXX Mail Mars Logistics Division**

**.NET Coding Exercise**

You wake up one morning to discover that Conde Monte Cristo and a selection of brave souls have successfully colonised Mars!XXX Mailhas partnered with Lurna to deliver parcels to the Martian colony via rockets launching from Starport Thames Estuary.

You're invited to complete the challenge that shows off your .NET and API skills.

**Parcel Tracking API (Mars Edition)**

XXX Mail now delivers to the single Martian city: New London, a geodesic colony near the base of Olympus Mons. Every delivery lands at a central hub and is handed off to AI-powered drones that complete final delivery within 1 hour. Your backend API must register parcels, track delivery progress, and support state transitions. Another team is responsible for the websitefrontend that will consume your API and another team will also consume your API for updates. This is the first iteration, an MVP, so it’s not expected to be feature-complete.

Optional: You may propose design changes or additional features to the system, e.g. how would you handle delays, simulate lost parcels occurring randomly or introduce basic filtering on the GET /parcels list endpoint as examples?If you do, briefly document your approach in the README.

**REST API Requirements**

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| POST | /parcels | Register a new parcel(with launch/arrival estimates) |
| PATCH | /parcels/{barcode} | Status updates (consumed by automated system) |
| GET | /parcels/{barcode} | Retrieve status + delivery history (consumed by website) |

**1. Register a Parcel**

**POST /parcels**  
Registers a new parcel for future launch and delivery.

**Request Example**

{

"barcode": "RMARS1234567890123456789M",

"sender": "Cuco Malanga",

"recipient": "Conde Monte Cristo",

"deliveryService": "Express",

"contents": "Signed C# language specification and a birthday card"

}

**Behaviour**

On creation, a parcel record should:

* Assign initial status: "Created"
* Set origin to "Starport Thames Estuary" and destination to "New London"
* Compute and store:
  + launchDate (based on selected delivery service)
  + etaDays
  + estimatedArrivalDate (based on launch + ETA)
* Use Earth UTC time throughout

**Delivery Services**

1. **Standard Orbital Delivery**
   * Launch every 26 months
   * Next standard launch date: 2025-10-01
   * ETA: 180 days
2. **Express Launch Service**
   * Launches on the first Wednesday of every month
   * ETA: 90 days

**Response Example**

{

"barcode": "RMARS1234567890123456789M",

"status": "Created",

"launchDate": "2025-09-03",

"etaDays": 90,

"estimatedArrivalDate": "2025-12-02",

"origin": "Starport Thames Estuary",

"destination": "New London",

"sender": "Cuco Malanga",

"recipient": "Conde Monte Cristo",

"contents": "Signed C# language specification and a Christmas card"

}

**2. Update Parcel Status**

**PATCH /parcels/{barcode}**  
Updates a parcel’s status to its next delivery stage.

**Request Example**

{

"newStatus": "OnRocketToMars"

}

**Valid Statuses**

* Created
* OnRocketToMars
* LandedOnMars
* OutForMartianDelivery
* Delivered
* Lost

Due to delivery costs, there is a strict no return policy.

**Valid Transitions**

| **Current Status** | **Allowed Transitions** |
| --- | --- |
| Created | OnRocketToMars |
| OnRocketToMars | LandedOnMars, Lost |
| LandedOnMars | OutForMartianDelivery |
| OutForMartianDelivery | Delivered, Lost |

**Terminal statuses:**Delivered, Lost

**Guidelines**

* If a PATCH is attempted with an invalid status change, return 400 Bad Request with an explanation.
* Use system time to enforce whether status changes are valid (e.g., launch must have occurred to become "OnRocketToMars").
* Suggestion: You may simulate time progression using dependency injection or a mockable clock for testability — explain your approach in the README.

**Response**

* 200 OK on success
* 400 Bad Request if invalid transition (with reason)

When a rocket launches, an external launch system will call PATCH on all relevant parcels with the corresponding launch date, transitioning them to OnRocketToMars.

Similarly, when the rocket lands on Mars and parcels are offloaded, another service will PATCH each one to LandedOnMars. When a drone is dispatched, a final PATCH updates to OutForMartianDelivery, and finally Delivered.

**3. Retrieve Parcel**

**GET /parcels/{barcode}**  
Returns full parcel details, including status and audit trail.

**Response Example**

{

"barcode": "RMARS1234567890123456789M",

"status": "Delivered",

"launchDate": "2025-09-03",

"estimatedArrivalDate": "2025-12-02",

"origin": "Starport Thames Estuary",

"destination": "New London",

"sender": "Cuco Malanga",

"recipient": "Conde Monte Cristo",

"contents": "Signed C# language specification and a Christmas card",

"history": [

{ "status": "Created", "timestamp": "2025-08-20" },

{ "status": "OnRocketToMars", "timestamp": "2025-09-03" },

{ "status": "LandedOnMars", "timestamp": "2025-12-02" },

{ "status": "OutForMartianDelivery", "timestamp": "2025-12-02" },

{ "status": "Delivered", "timestamp": "2025-12-02" }

]

}

**Barcode Format**

Your system must validate barcodes using this format:

RMARS1234567890123456789M

* Each parcel has a unique barcode
* Starts with RMARS
* Followed by 19 digits (0-9)
* Ends with an uppercase letter (A–Z) [strictly speaking would be a checksum digit, but details omitted for brevity, so you only need to know it’s a capital letter]
* Invalid barcodes → return 400 Bad Request

**Sample Happy Path: End-to-End Example**

**Step 1 – Parcel Creation (POST /parcels)**

{

"barcode": "RMARS1234567890123456789M",

"sender": "Cuco Malanga",

"recipient": "Conde Monte Cristo",

"deliveryService": "Express",

"contents": "Signed C# language specification and a Christmas card"

}

**Response**

{

"barcode": "RMARS1234567890123456789M",

"status": "Created",

"launchDate": "2025-09-03",

"etaDays": 90,

"estimatedArrivalDate": "2025-12-02",

"origin": "Starport Thames Estuary",

"destination": "New London",

"sender": "Cuco Malanga",

"recipient": "Conde Monte Cristo",

"contents": "Signed C# language specification and a Christmas card"

}

**Step 2 – Update to OnRocketToMars (PATCH)**

{

"newStatus": "OnRocketToMars"

}

Response: 200 OK

**Step 3 – Update to LandedOnMars (PATCH)**

{

"newStatus": "LandedOnMars"

}

Response: 200 OK

**Step 4 – Update to OutForMartianDelivery(PATCH)**

{

"newStatus": "OutForMartianDelivery"

}

Response: 200 OK

**Step 5 – Final delivery(PATCH)**

{

"newStatus": "Delivered"

}

Response: 200 OK

**Step 6 – GET /parcels/{barcode}**

This can be called at any time. Below is an example after completion of the delivery with the full history.

{

"barcode": "RMARS1234567890123456789M",

"status": "Delivered",

"launchDate": "2025-09-03",

"estimatedArrivalDate": "2025-12-02",

"origin": "Starport Thames Estuary",

"destination": "New London",

"sender": "Cuco Malanga",

"recipient": "Conde Monte Cristo",

"contents": "Signed C# language specification and a Christmas card",

"history": [

{ "status": "Created", "timestamp": "2025-08-20" },

{ "status": "OnRocketToMars", "timestamp": "2025-09-03" },

{ "status": "LandedOnMars", "timestamp": "2025-12-02" },

{ "status": "OutForMartianDelivery", "timestamp": "2025-12-02" },

{ "status": "Delivered", "timestamp": "2025-12-02" }

]

}

**What to Submit**

* Code (GitHub repo or zipped folder)
* .NET 8 or later
* Use in-memory data or a simple JSON file (no database needed)
* README with:
  + Setup instructions (`dotnet run`, `dotnet test`, ensure Swagger available at `/swagger`)
  + Summary of design choices and trade-offs
  + A walkthrough of your solution and what you might improve for enterprise scale?
  + Explanation of any design choices, assumptions, or shortcuts (shortcuts allowed in the interest of time)
  + AI tool usage (see below)
* Your API should expose a Swagger UI page (OpenAPI) so we can easily inspect and test your endpoints.

**Evaluation Criteria**

| **Area** | **What We Look For** |
| --- | --- |
| **API design** | Clear routing, separation of concerns |
| **Logic correctness** | Time-based logic, status transitions, barcode validation |
| **Testing** | Meaningful automated tests covering core logic |
| **Code quality** | Readable, testable, maintainable code |
| **Communication** | Good commit hygiene, clear README |
| **AI Use (if any)** | Statedin README |
| **Swagger/OpenAPI** | Enabled at /swagger, auto-generates docs |

**AI Usage Guidelines**

You may use Copilot, ChatGPT, or similar tools for **supporting tasks**, such as:

* Documentation and setup instructions
* Generating or scaffolding automated tests
* Creating test or seed data
* Idea generation or exploring approaches

Please **avoid** using AI for:

* Full scaffolding of the application (e.g. generating controllers, services, and models in one go)
* Core application logic (e.g. status transitions, time/date handling, validation rules)

We want to assess **your own engineering skills**, not just your tool usage. We’ll review your code in the interview, so please be ready to explain and justify your key decisions.

In your README, include:

* Which AI tools you used
* What you used them for
* How you validated or adapted any generated code (if applicable)

**Time Expectation**

Suggested target time is**2–3 hours**.  
This is an MVP. You do not need to build a full production system.We value **code quality over completeness** — it's fine to make trade-offs and shortcuts, but please document them.Be especially careful with date/time handling and barcode validation — clarity and correctness matter.

Have fun — and good luck. The future of tracking Martian deliveries may depend on you!