Michael API Breaking Changes Answers

1) **What Is a Breaking Change?**
   A breaking change occurs when an update to the API disrupts existing consumers who rely on the current schema or behavior. For example:

1. Renaming or removing properties – Changing "Weather" to "Forecast" or removing it entirely would cause deserialization to fail in frontend apps expecting "Weather".
2. Changing the response format – Switching from JSON to XML, or altering the data type of a field (e.g. "temperature" from string to number) without clear migration guidance.
3. Modifying the structure or endpoint – Changing the URL path (/api/weather to /api/forecast) or nesting the hourly data deeper under another property without backward compatibility.

2) **Coordinating Across Multiple Frontends**
   When some clients update frequently and others lag behind by months, it's crucial to support versioning and phased transitions:

- Introduce versioned endpoints (e.g. /api/v1/weather, /api/v2/weather) and avoid changing the behavior of live production versions.
- Deprecation strategy: Clearly communicate deprecation timelines to consumers and provide migration guides.
- Compatibility testing: Maintain contract tests or consumer-driven pact tests to validate backward compatibility across versions.

3) **How to Catch Breaking Changes During Development**
   Here's how to catch issues before they hit production:

- Contract tests and CI/CD enforcement: Ensure schema validation and test suites (unit, integration, E2E) run on every pull request.
- Schema diffing tools: Automate checks on OpenAPI (Swagger) specs to flag any breaking changes between commits.

4) **Policy for Releasing Changes**

- Versioned releases: Each breaking change results in a new API version; minor changes follow semantic versioning.
- Environment gates: Every build moves through dev → stage → production, tested and validated at each level.
- PR Review Standards: Code changes must include test coverage and PRs review by seniors and leads.
- Release Communication: Publish changelogs, notify consumers via Slack/email, and document mock samples for frontend devs.
- Progressive rollout: Use a load balancer or API gateway to gradually shift traffic from v1 to v2 and monitor error rates.