- Code Design
    - Domain driven design using Domain Models
    - Modular structure for code management e.g. easy to add remote routes
    - Use of middleware such as Error Handlers, Validation functions to reduce repeatable code.
    - Static Typing using Typescript
- Quality
    - Clear and understanding with familiar naming conventions, typings
    - Test Driven Development to ensure all the functional tests are passing as you make changes
    - Linting to ensure good code standards and well formatted code
    - Clear separation of helpers, constants for ease of use.
- Error Handling
    - Graceful handling of errors using validation errors, conflict errors and not found errors.
    - Logging of Error for tracing.
    - Use of correlationId for debugging and/or tracing
- Code Performance
    - Use of custom secondary indexing in the database for optimized query support.
- Security
    - Use of helmet middleware (https://www.npmjs.com/package/helmet) to ensure best security practices in building secure RESTful APIs. This includes preventing cross site scripting(xss filter), content security policy (csp) etc.
    - Use of logger and corelationId field for tracing and debugging

- Infra/Operability
    - Cloud native support using Docker that can be easily deployed as containers.
    - Can be deployed as Microservices with support of health endpoints to ensure the health of the service.
    - Support for environment variables to allow configuration such as server url and/or port or database url and/or port. This can be used in conjunction with Kubernetes config files for example.
    - Furthermore the logs can be forwarded to Observability tools such as New Relic or splunk using agents.

- Documentation
    - Comments provided for ease of understanding of code.
    - Tests are self explanatory towards the expected feature and behaviour.
    - Validation rules using Joi are easy to understand, reuse and extend.