Rental Agreement Implementation Requirements

This document augments the functional specification provided by program management. Please refer to that document for additional details.

This document is broken into two sections. The first will be for the initial prototype. The second will note future considerations to take into account while coding the prototype. Future considerations will not be developed for the initial prototype.

With this document and the functional specification, engineering should be able to successfully build the prototype.

# Assumptions

The solution is expected to be one of several micro services deployed in a managed environment such as Kubernetes or a cloud solution like AWS Fargate using Docker containers to manage service dependencies. It is assumed that this service only handles rental agreement generation and is not required to track tool availability as indicated via the functional specification.

# Prototype

## Implementation Requirements

* Functionality will be delivered as a stateless micro service implemented using Java Spring/SpringBoot Model View Controller (MVC) Web Server.
* Maven for package management and building.
* Logging will be done to standard out.
  + Useful to utilize in Kubernetes or cloud deployments and allows for log aggregations to be used easily.
  + Log levels will be supported and used as needed
    - ERROR
    - WARN
    - INFO
    - DEBUG
  + Implement the sprint actuator for logging to provide dynamic log level changes.
    - That will improve debugging capabilities.
* See the API Specification for REST details.
  + Filename: rental\_agreements.yaml
  + Versioning will be handled via URL. That makes it obvious which version you are attempting to use as a consumer. Other methods are more obscure and easier to make a mistake with.
  + Endpoints should use dashes to separate multiple words.
* Setup Data Transfer Objects (DTOs) in three categories creating a separation of concerns and ease of maintenance.
  + Request
  + Response
  + Model
* Assume all dates are UTC.

## Non-functional Requirements

* Follow standard coding guidelines for the company. Refer to the internal documentation for those guidelines.
* Unit tests for internal method testing
* Functional tests to exercise REST endpoints
* Portability is handled via Java

# Future Considerations

* User authentication and authorization will need to be added.
  + Audit logging will then be required.
* Log formatting should be updated to match company guidelines.
* Rate limiting and other security items should be addressed for the endpoints.
  + See company guidelines for those details.
* The service will not use hard coded item details and pay rates. They will be pulled from an external source.
* Holidays will also be derived from an external source and not hard coded.
* Log aggregation tools will be used for long term storage of logs.
* Endpoints should only allow TLS communication.
* Add a /health endpoint
  + Will show service status and all dependencies (their info endpoint)
* Add an /info endpoint
  + Will only show service status
* Items like availability, scalability, reliability can only be addressed with additional knowledge of the deployment and operation of the service in the broader environment.
* Add token to track transactions through the system and add to each message logged when handling a transaction.
* Sensitive data should not be logged. Ex. Passwords and any PII.
* Handle local timezones.