pink: not done yet green: met yellow: not entirely met blue: not sure if met red: not met at all

Requirements model	**8**
Functional requirements are properly formulated	2
Non-functional requirements are properly formulated	1
Requirements are identifiable	1
Requirement specifications systematically follow a template (e.g., ID, user stories, etc.)	2
Requirements registered as issues in project backlog	1
Requirements are verifiable	1
Use Case Diagram	**7**
UC diagram is syntactically well-formed	1
UC diagram is easy to understand (e.g., not a single messy diagram with all UCs)	1
Use cases are properly named (Verb + Subject)	1
Actors are properly named (Roles played in the system)	1
External subsystems included as actors	1
UC-Actor assignment is semantically meaningful	2
Detailed use case specifications	**5**
UC specifications are numbered hierarchically	1
Main flow for UCs is defined	1
Alternate flows for UCs are defined	2
Steps in scenarios are systematically formulated (Who does what?)	1
Domain Model	**15**
Class diagram of domain is easy to understand	1
Rationale of key decisions are documented	1
Classes are consistently used	4
Attributes are consistently used	2
Enumerations are consistently used	1
Generalization hierarchy is properly used (incl. abstract classes)	1
Containment hierarchy is elaborated (is there composition? not flat hierarchy?)	1
Multiplicities are appropriate (e.g., not too generous, not too restrictive)	1
Associations are appropriate (e.g., not only bidirectional)	3
Persistence Layer	**15**
JPA classes are compliant with domain model in UML	3
JPA annotations are consistently used	2
DAO implementation exists for CRUD methods of classes (auto generated or manually written) (Needs to be evaluated separately for each class)	10
Testing of Persistence Layer	**15**
Read test cases exist for each class	5
Write test cases exist for each class	5
Test suite demonstrates that application can read and write - objects	1
Test suite demonstrates that application can read and write - attributes	1
Test suite demonstrates that application can read and write - references	1
Database contents cleared / reverted after test method	2
Build system	**10**
Testing documentation (testing summary, required configs)	3
The project builds locally with Gradle (gradle build -xtest or ./gradlew build -xtest)	3
Tests run and pass locally	4
Project Management and Project Report	**20**
Project deliverable provided on wiki	2
Backlog is maintained in GitHub Projects	2
Issues are created at the beginning of sprint	2
Issues are assigned to milestones	2
Issues are assigned to responsible person	2
Lifecycle of issues is continuously managed in GitHub	2
Project Report has welcome page (introduction to group, scope of project)	2
Documentation of team operation (minutes of meeting recorded, design decisions recorded)	2
Individual roles are detailed	2
Individual efforts are summarized in the table	2
Code style	**5**
There are comments for persistence layer methods	1
There are comments for nontrivial code blocks	1
Code is free from unused variables and unused imports	1
Methods are not too long (approx. 20 lines)	1
Naming of variables, methods follow conventions	1
TOTAL	**100**

<-- we might need to restructure to make it more visually appealing