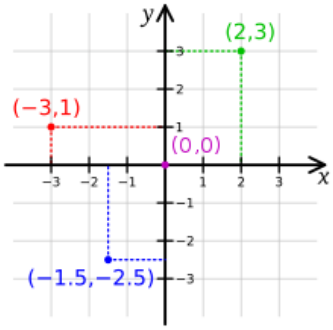
MTAT.03.295 – Agile Software Development
Regular Exam – 15 December 2020
Part 2. Practice

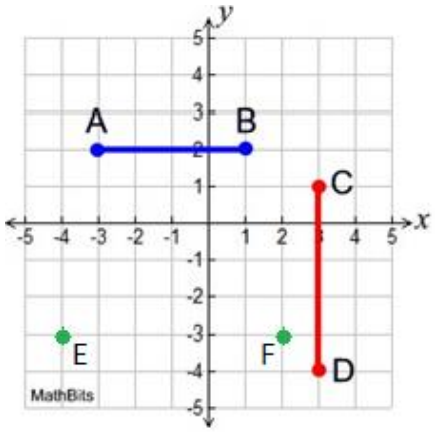
Notes:

- You are can access any resource on the web
- You are not allowed to communicate with anyone during the exam in any way (except with the lecturer).
- You MUST create a **PRIVATE** repository at https://gitlab.cs.ut.ee, and add the teaching staff there as collaborators (with admin rights). You should submit a TXT or PDF file with the link to your repository with the solution. Please leave your solution in the "master" branch of your repository and do not make any commits to "master" after the end of the exam.
    - Username: orleny85, email: orlenys.lopez.pintado@ut.ee.
    - Username: scott, email: ezequiel.scott@ut.ee.
- **IMPORTANT**: You MUST commit to the repository each time that you complete a task or requirement in the exam. You can find the minimal list of commits messages that we expect. You are welcome to add other commits, for example, if you fix a task completed before. For each commit missing, you will get a penalization of 20% of the points received corresponding to the task. Note that commits whose messages do not correspond to the task will get a penalization of 20% too. For example, if your commit message says, "Task 2, Completed", be sure that the code submitted corresponds to task 2; however, other parts of the implementation may be updated too. Finally, we will discard all the commits submitted after 12:00 (EEST time).
- **IMPORTANT**: We strongly suggest you not to share your code before the end of the exam, to avoid misunderstandings. In the case of projects in which it is evident that the students committed fraud, all the involved students will get 0 points in the exam, meaning that they all fail it. Also, if we observe suspicious behavior either in the implementation or the commits, the students involved will be called for an online interview with the teaching staff. Examples of suspicious behavior can be, all the commits happening at the end of the exam or with a non-realistic timeline, too many similarities between projects, etcetera. During the interview, the students will be asked about the code they submitted. Note that, during the interview, we will downgrade the mark of the student for each question not answered correctly, meaning that the student may fail the exam in the meeting.

Geometry

Our team has been hired to implement a system for high school students to study math. For the first release of the software, we will implement a Phoenix application with a basic set of features to handle points and line segments (see figure below).





The application must satisfy the following requirements:

- **R1.** A point in the plane is defined by a ***name*** and two ***coordinates***: two real numbers, e.g., x = 3.5, y =2.25. Thus, the system must allow a user to introduce a point's ***name*** and its ***coordinates***, which should be stored in the database (if they are valid). The system must guarantee the ***coordinates*** of a point provided by a user are valid, i.e., not empty values, and they must be real numbers. Besides, the system must guarantee that there are no points with the same ***name*** in the database, i.e., each point's ***name*** must be non-empty and unique.
- **R2.** A line segment in the plane is defined by two points. The system must allow a user to create line segments from existing points in the database. To that end, the system must allow a user to introduce the ***name*** of two points, then verify both points exist in the database. Otherwise, an error message must be shown to the user. If both points exist, the system must create a new segment and update the database accordingly. Besides, the system must calculate the new segment's longitude[1] and display a message notifying a user that a new segment was created. That message must also include the name of the points, the segment's longitude, and how many points have a smaller longitude than the new one. For example, consider that the points ***A***, ***B***, ***C***, ***D***, ***E***, and ***F*** (in the figure above) exist in the database and the segments ***A-B*** and ***C-D***. Then, the user requests to create the segment joining the points ***E*** and ***F***. Accordingly, the database must be updated. Besides, the system must show the following message to the user: "*The segment E-F of longitude 6 was created successfully. The new segment longitude is greater than 2 existing segments in the system*".

The task breakdown and corresponding marks are as follows:

[1] To calculate the longitude of a segment formed by two points (x1, y1) and (x2, y2) you can use the following formula $\sqrt[2]{(x1-x2)^2+(y1-y2)^2}$

- ***Task 1 (4 points)*** [BDD]: Specification of a Gherkin user story describing the requirement R2. As a minimum, your feature must include the clauses ***Given***, ***And***, ***When***, and ***Then***. Your user story must specify that some points and segments exist in the databases (providing a table with their data). Followed by the information required to create a new line segment and the action which triggers such creation. Finally, the corresponding message to display to the user, including the involved points' names, the segment's longitude, and the number of line segments with lower longitude. You must consider only the scenario in which the segment is successfully created.
- ***Task 2 (4 points)*** [BDD – steps implementation]: Implementation of the white_bread steps described by the user story in Task 1.
- ***Task 3 (2 points)***: Setup of the application routes to support the creation of the two different resources described above: points and segments.
- ***Task 4 (6 points)***: Setup & implementation of models (via migrations) and seeding the database with some initial data based on your models. The seeding of the database must include both points and segments.
- ***Task 5 (12 points)***: Implementation of controllers. You must implement the operations to render the templates to provide the input data for points and segments. You must also implement the operations regarding the validation/creation of new points and line segments, including the messages to notify the creation/rejection (as described in requirements R1 and R2).
- ***Task 6 (4 points)***: Implementation of the views and templates to introduce the input data for creating points and line segments. You do not need to create/render a new template to display the messages. For that, you can just write in the flash as we did during the course practical sessions.
- ***Task 7***: (TDD) Unit tests for either models/controllers checking the requirements R1 and R2. A single test may include several requirements. Specifically, your tests must consist of at least:
    - ***R1 (3 points)*** – A negative case in which the name of a point is not unique. Thus, the system displays an error. One positive case in which the input is valid, i.e., including coordinates and name, the new point is created, and the database is updated successfully.
    - ***R2 (5 points)*** – A negative case in which at least the name of the points provided to create a segment does not exist. Thus, the system displays an error. Also, a positive case, in which the line segment is created successfully. The test must check that the line segment's longitude and the number of points with lower longitudes are calculated correctly.

Minimum Commits – Messages (NOT necessarily in the same order). Note that each message must correspond to a different commit.

- Initial Commit
- Task 1 Completed
- Task 2 Completed
- Task 3 Completed
- Task 4 Completed
- Task 5, R1 Completed
- Task 5, R2 Completed
- Task 6 Completed
- Task 7, R1 Completed
- Task 7, R2 Completed