**HOMEWORK WEEK 5-6**

Madina Baimaganbetova

**TASK 1 (Agile Techniques)**

**Question 1**

**Complete definitions for Scrum related key terminology provided below**.

SCRUM CEREMONIES

**Product backlog refinement** - An activity that is carried out by the Product Owner with the participation of all team members. Includes adding details, evaluating and arranging items in the Product Backlog. Backlog refinement usually takes no more than 10% of the Scrum Team's time in a Sprint.

**Sprint planning** – This is the meeting that initiates the Sprint and plans the work for that Sprint. It is one of the 5 Scrum Activities. The entire Scrum Team is involved in Sprint Planning, and other people may be invited to consult. The following questions are to be answered as a part of sprint planning

1) Why is this Sprint valuable?

2) What exactly can be done in this Sprint?

3) How the selected work will be carried out?

**Daily scrum** - One of the 5 Scrum Activities. This meeting lasts no more than fifteen minutes and is held every working day in the same place at the same time. All developers participate in it. It provides information to measure progress and highlights obstacles. As a result, developers may come to the need to reschedule work within the Sprint.

**Sprint review** - One of the 5 Scrum Activities. Conducted at the end of the Sprint for customers and stakeholders to review and provide feedback on the Increment and for the Scrum Team to adapt the Product Backlog if necessary. For a Monthly Sprint, the Sprint Review lasts a maximum of 4 hours.

**Sprint retrospective** - One of the 5 Scrum Events. The Sprint Retrospective gives the Scrum Team an opportunity to review their work and create an improvement plan for the next Sprint. The Retrospective takes place after the Sprint Review, before the Sprint Planning. For a Monthly Sprint, this meeting is limited to 3 hours.

SCRUM ROLES

**ScrumMaster** - This is one of the 3 areas of responsibility in the Scrum Team. The Scrum Master is the Servant Leader for the Scrum Team and for the organization. Helps everyone understand Scrum theory and practices, trains the team to remove barriers, coaches the team, and facilitates Scrum Events. In fact, he is the owner of the process, responsible for the effective work of the Scrum team.

Scrum Master responsibilities can be divided in three major groups basing on their relevance for the team, product owner and organization:

Scrum Team:

• coaches team members to develop self-management and cross-functionality.

• helps the team focus on creating High Value Increments that meet the Done Criteria;

• contributes to the removal of obstacles hindering the progress of the Scrum Team;

• makes sure that all Scrum Activities are ongoing, positive, productive, and within time limits.

Product Owner:

• helps to find techniques for effectively defining the Product Goal and managing the Product Backlog;

• helps to understand the need for clear and concise elements of the Product Backlog;

• helps to apply empirical product planning in a complex environment;

• Facilitates interaction with stakeholders upon request or as needed.

For the Organization as a whole:

• guides, trains and coaches the organization in all aspects of Scrum;

• plans the transition to Scrum, participates in the implementation and advises on the use of Scrum in this organization;

• helps staff and stakeholders understand and apply an empirical approach to integrated work;

• removes barriers between stakeholders and Scrum teams

**Product Owner** - This is one of the 3 areas of responsibility in the Scrum team. The Product Owner is responsible for maximizing the value of the product resulting from the work of the Scrum Team. His responsibilities also include curating and prioritizing the Product Backlog. About 50% of the time the Product Owner spends with customers and stakeholders, the remaining 50% is shared with the team. As responsible for the effective management of the Product Backlog, the Product Owner:

• develops the Product Purpose and makes it explicit;

• creates and clearly explains the Product Backlog Elements;

• organizes the Product Backlog Items;

• ensures transparency, accessibility and understanding by all of the Product Backlog.

The Product Owner can perform this work himself or delegate its performance to other people, remaining responsible for its result.

For the Product Owner to succeed, the entire organization must respect their decisions. These decisions are reflected in the content and order of the Product Backlog items, as well as in the Increments that are reviewed during the Sprint Review.

it is one person, not a committee. The Product Owner can reflect the needs of many stakeholders in the Product Backlog. Those wishing to change the Product Backlog can do so by trying to convince the Product Owner.

**Development Team** - This is one of the 3 areas of responsibility in the Scrum Team. Developers are the people who work on the Sprint Backlog Items. They have all the necessary competencies to create a working Product Increment every Sprint.

Developers:

• create the Sprint Backlog;

• daily adapt their plan to achieve the Sprint Goal;

• strive for quality through compliance with the Readiness Criteria;

• help each other develop their professional skills necessary for product development.

The term "Developers" in Scrum is not related to the narrow meaning of the word "developers" common in the business environment, meaning programmers and other engineers involved in software development.

Even if an IT product is being created, from the point of view of Scrum, Developers include people who have positions of analysts, quality engineers (testers), designers, technical writers, etc.

In the development and support of products that are loosely or not at all related to software development, Developers may include anyone who contributes to the Increments. For example, marketers will be included as Developers in the team involved in the development and support of tariffs for the company's services.

**Question 2**

You are leading a development team that was given a task to create a new yoga booking system.

High level description of the system is as follows:

· It has a very simple interface to **accept user input (bookings)** and **display classes information**

· All bookings, appointments, schedules etc should be stored in a SQL database.

· There is a ‘backend’ system that should be written in Python to handle the logic and manage the data flow.

Your team has two weeks to build a simple prototype that will be shown to the client to seek their feedback and discuss further enhancements.

TASK

· Break this task into smaller stories (chunks of work) for the team to work on.

· Assume that one person works on one task.

· Mark tasks that can be worked on in parallel and perhaps those that need to be worked on in particular order.

**Solution:**

From **user side**, system will have the following functionality:

1. Display class info
2. Select class
3. Display class schedule with availability
4. Book class
5. Payment page
6. Pay, accept payment
7. Payment, booking notification via email (with cancellation link)
8. Class notification via email (X hours prior class)
9. Class reschedule/cancellation/refund via link in email (see 7)

Please refer to the scheme below:

Diagram

Description automatically generated

There will be **4 microservices** available on this stage with following subtasks:

1. Booking service
   * Add booking (get user data)
   * Remove booking
   * Reschedule booking
2. Payment service
   * Accept payment
   * Revert payment
3. Notification service (via e-mail)
   * Booking notification (book, cancel, reschedule)
   * Class reminder notification
   * Class review (will be optional)
4. Main page
   * Display class info
   * Class schedule/availability
   * Display class reviews (optional)

**Each of these services can be developed in parallel** because we can mock other services’ outputs. Finally, all the services will be brought together and real inputs/outputs will be used for final tests.

**Each task within the service can also be developed in parallel**, as we are planning to use mock inputs.

The yoga classes database will contain **4 tables** at this stage, including Classes table, Bookings table, Users table, and Payments table. Moreover, Reviews table can also be added (optional)

|  |  |
| --- | --- |
| **Classes** | |
| Primary key | class\_id |
|  | description |
|  | capacity |
|  | level |

|  |  |
| --- | --- |
| **Bookings** | |
| Primary key | booking\_id |
| index | class\_id |
| Second\_index | datetime |
|  | User\_id |
|  | Payment\_id |
|  | Status (completed/  cancelled/  rescheduled) |

|  |  |
| --- | --- |
| **Payments** | |
| Primary key | payment\_id |
|  | Booking\_id |
|  | User\_id |
|  | datetime |
|  | amount |

|  |  |
| --- | --- |
| **Users** | |
| Primary key | user\_id |
|  | e-mail |
|  | First name |
|  | Last name |

|  |  |
| --- | --- |
| **Reviews (optional)** | |
| Primary key | review\_id |
|  | class\_id |
|  | User\_id |
|  | datetime |
|  | review |

**TASK 2 (SQL)**

***Question 1***

**Design a cinema booking system.**

Think how you would approach the problem and what are potential ways of solving it

You do not need to write actual code, but describe the high-level approach:

· Draw a list of key requirements

· What are your main considerations?

· What would be your common or biggest problems?

· What components or tools would you potentially use?

· You are welcome to draw a diagram (a very simple one) for the process flow to explain how it is going to work.

**Solution:**

My booking system considers individual cinemas, not cinema multiplex like ODEON etc. Therefore it has not option of selection different cinemas across the city.

From **user side**, system will have the following functionality:

1. Display ongoing film info
2. Select film
3. Display film schedule with seats availability
4. Book movie ticket, select seats
5. Payment page
6. Pay
7. Payment, booking notification via email (with cancellation link)
8. Ticket return/refund via link in email (see 7)

The system from user side will look like following:

Diagram

Description automatically generated

The system should comply with the following functional requirements

**Key requirements:**

1. Add/remove films to database
2. Display current films
3. Book tickets
   * Buy
   * Refund
   * Select place
4. Payment system
   * Accept payments
   * Revert payments
5. Notification system via e-mail
   * Confirm booking/ send tickets

There will be **5 microservices** available on this stage with following subtasks:

1. Booking service
   * Add booking, select seats (get user data)
   * Remove booking
2. Payment service
   * Accept payment
   * Revert payment
3. Notification service (via e-mail)
   * Booking notification (book, cancel, reschedule)
4. Main page
   * Display currently ongoing movies
   * Display movie info
   * Movies schedule/ seats availability
   * Display movie reviews from IMDB, rotten tomatoes
5. Cinema repertoire service:
   * Add new ongoing movie
   * Remove old movies

**Each of these services can be developed in parallel** because we can mock other services’ outputs. Finally, all the services will be brought together and real inputs/outputs will be used for final tests.

**Each task within the service can also be developed in parallel**, as we are planning to use mock inputs.

The cinema booking database will contain Movie info table, Movie schedule table, Cinema Room table, Bookings table, Users table, and Payments table. Moreover, Reviews table can also be added (optional)

|  |  |
| --- | --- |
| **Movies info** | |
| Primary key | movie\_id |
|  | description |

|  |  |
| --- | --- |
| **Movies schedule** | |
| Primary key | Movie\_viewing |
|  | movie\_id |
|  | Cinema\_Room\_id |
|  | datetime |

|  |  |
| --- | --- |
| **Cinema room** | |
| Primary key | room\_id |
|  | capacity |
|  | Seat\_schema |

|  |  |
| --- | --- |
| **Bookings** | |
| Primary key | booking\_id |
| index | viewing\_id |
| Second\_index | datetime |
|  | User\_id |
|  | Payment\_id |
|  | Status (completed/  cancelled) |

|  |  |
| --- | --- |
| **Payments** | |
| Primary key | payment\_id |
|  | Booking\_id |
|  | User\_id |
|  | datetime |
|  | amount |

|  |  |
| --- | --- |
| **Users** | |
| Primary key | user\_id |
|  | e-mail |
|  | First name |
|  | Last name |

**What are your main considerations?**

1. System should be fast, not laggy
2. System should be able to handle big traffic, especially in cities with big population
3. System should protect user privacy and sensitive information (payment info, user info)
4. System should be easy to use, user friendly

**What would be your common or biggest problems?**

1. Double booking / concurrent booking streams
2. DDOS attacks / rate limiter
3. Reserving the seat for limited amount of time until payment
4. Malicious overbooking

**What components or tools would you potentially use?**

1. External payment system (SaaS)
2. REST API
3. External film details API (IMDB, Rotten Tomatoes)
4. SQL database
5. Python backend framework (Flask)
6. Containerized microservices (Docker, Kubernetes)
7. Cloud providers (AWS, GCP)
8. Notification delivery system (SaaS)