# بسم الله الرحمن الرحیم

## تمرین شماره سه درس مهندسی نرم‌افزار

## گروه شماره ۴


**انجام دهندگان:**

**سپهر رفیعی – ۸۱۰۱۹۶۶۷۹**

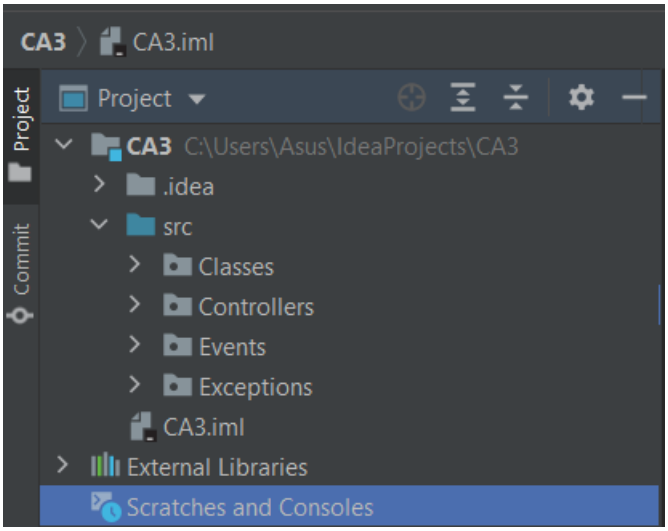
**محمد معین شفی – ۸۱۰۱۹۶۴۹۲**

**محمد سعید کودک پور – ۸۱۰۱۹۶۵۴۰**


**بهار ۱۴۰۰**

# پرسش یکم

در ابتدا لازم است کمی در مورد معماری و تصمیماتمان برای پیاده سازی این سوال صحبت کنیم. برای این قسمت، ما از معماری MVC بهره گرفته ایم. هرچند در این تمرین، نیازی به پیاده سازی فرانت نیست و ایجاد UI برنامه مورد نظر ما نیست. فریمورک MVC یا همان Model-View-Controller، یک الگوی معماری است که اپلیکیشن را به سه جز اصلی تقسیم می کند. هر کدام از این اجزا برای مدیریت جنبه های خاص توسعه نرم افزار ساخته می شوند. اولین جز Model است که مربوط به منطق داده ای برنامه است که کاربر با آن ها کار می کند. این قسمت داده هایی که بین Controller و View جا به جا می شود، است. جز دوم View است که برای منطق UI برنامه استفاده می شود. در این تمرین این مورد را نیاز نداریم. و جز آخر، Controller است که تمام منطق بیزنس و requestهایی که به برنامه می آید را هندل می کند و داده ها را با مولفه Model دستکاری می کند و نتیجه نهایی را تولید می کند. همچنین قرار است از الگوی event sourcing استفاده کنیم، پس در این صورت نیاز به یک سری event هم داریم. پس در کل یک سری کلاس اصلی برای موجودیت های اصلی داریم، یک سری Controller برای مدیریت درخواست ها، یک سری event برای ذخیره سازی استیت اپلیکیشن و ذخیره سازی تمامی تغییرات و در نهایت یک سری اکسپشن خواهیم داشت. در نهایت چیزی شبیه به زیر خواهیم داشت:




حال به سراغ قسمت اصلی این تمرین برویم. از آنجا که گفته شده در شروع انتخاب واحد، کلیه داده های مورد نیاز، از سرویس های دیگر گرفته می شود و در قالب داده ساختارهایی ذخیره می شود، یک کلاس دیتابیس داریم که یک instance از دیتابیس اصلی می گیرد و در واقع کلیه داده های مورد نیاز از سرویس

های اصلی گرفته می شوند. در واقع، اطلاعاتی که باید از سرویس ها گرفته شوند و در حافظه اصلی نگه داری شوند، شامل موارد زیر هستند:

- اطلاعات دانشجویان نظیر شماره دانشجویی، رشته، معدل، تعداد واحدهای گذرانده شده و ... در حافظه اصلی نگه داری می شوند.
- اطلاعات هر درس و هر ارائه آن گرفته شده و در حافظه اصلی ذخیره سازی می شوند. اطلاعات درس شامل مواردی چون کد درس، واحد درس، زمان امتحان و لیست درس های پیش نیاز خواهد بود و هر ارائه درس، اطلاعاتی نظیر، کد کلاس، استاد، زمان و روز های کلاس، ظرفیت درس و لیست انتظارش، دانشجویان ثبت نامی و در لیست انتظار، وضعیت باز یا بسته بودن درس و ... خواهد بود.
- اطلاعات استاد و کارشناس آموزشی هم در حافظه اصلی ذخیره می شوند.
- در نهایت اطلاعات درس اخذ شده توسط دانشجو هم نگه داری می کنیم. این که می خواهد درس را حذف کند یا خیر، در لیست انتظار است یا خیر، درس برایش نهایی شده است یا خیر را نگه داری می کنیم.

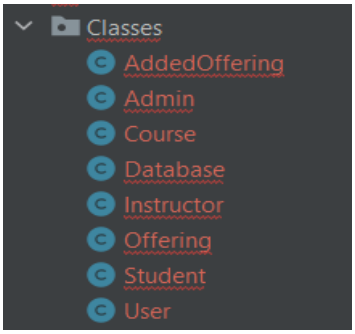
در نهایت یک سری Event خواهیم داشت تا تغییرات و استیت اپلیکشن را ذخیره سازی کنیم. این Event ها شامل موارد زیر هستند:

یک ایونت پایه داریم که شامل کد ایونت، زمان اتفاق افتادن آن، تایپ آن که چه نوعی است، یوزری که آن را فراخوانی می کند، خواص آن در قالب json، valid بودن یا نبود آن و .... را دارد. سایر event ها از آن ارث بری می کنند. ایونت های اصلی زیر را داریم:

*اولین Event، برای دیدن لیست ارائه های مختلف درس است. دومین Event برای زمانی است که یک enrollment به درسی انجام می شود. سومی برای حذف آن enrollment است، چهارمی برای تغییر ظرفیت درس است. پنجمی و ششمی برای تغییر زمان امتحان و زمان کلاس درس هستند. هفتمی برای وضعیت درس است که آیا می شود در آن ثبت نام کرد یا خیر. دو تای آخر هم برای اضافه شدن یا حذف شدن در لیست انتظار می باشند.*

سپس یک سری کنترلر برای هندل کردن ریکوئست و فهمیدن نوع ریکوئست داریم که در آن ها بعد از یافتن نوع درخواست، تابع مورد نظر را صدا می زنیم و خطاهای احتمالی را چک می کنیم.

حال کد ها را با هم ببینیم. در ابتدا کد موجودیت هایی که از دیتابیس می گیریم و در حافظه اصلی ذخیره می کنیم را ببینیم:




## 1)User:

این مورد یک سری اطلاعات پایه ای برای تمامی افراد نگه داری می کند که طبق role خودشان از این کلاس ارث بری می کنند:

```
public class User {

   private String firstName;
   private String lastName;
   private Integer phoneNumber;
   private String email;
   private String information;
}
```

## 2)Admin:

این مورد همان کارشناس آموزش است. علاوه بر اطلاعاتش یک سری تابع برای تغییر ظرفیت درس، تغییر وضعیت باز یا بسته بودن درس، تغییر تایم کلاس یا امتحان و دریافت اطلاعات درس ها دارد. همگی توابع با کامنت توضیح داده شده اند. همچنین برای هر تابع موردنظر، Event مربوطه هم صدا زده می شود که چون نیازی به پیاده سازی توابع نیست دیگر نیاورده ایم. برای مثال هنگام فراخوانی changeCapacityOfCourse، ایونت ChangeCourseCapacityEvent که در ادامه می بینیم، صدا زده می شود تا تغییرات را نگه داری کنیم.

```
public class Admin extends User{

   private String adminId;
   private String role;

   public void changeCapacityOfCourse(Offering offering, Integer
```

```
capacity) throws CourseNotFoundException {
        //change capacity of the specified course
    }

    public void changeCourseChosenStatus(Offering offering) {
        //Specify if the course can be taken or not and change
canBeChosen in offering
    }

    public void ChangeClassTimeOfCourse(Offering offering) {
        //Change class time of a course
    }

    public void ChangeExamTimeOfCourse(Offering offering) {
        //Change exam time of a course
    }

    public String GetCourseInformation(Offering offering) {
        //Get information of the specified course
    }

    public ArrayList<String> GetAllCoursesInformation() {
        //Get information of all courses
    }
}
```

## 3)Student:

این مورد، دانشجو است که علاوه بر اطلاعاتش، یک تابع برای اضافه کردن درس به برنامه اش دارد. در این تابع تمامی محدودیت ها چک می شود. محدودیت هایی اعم از اینکه درس را قبلاً نگذرانده باشد یا همین درس را دو بار به برنامه اش اضافه نکند، اینکه تعداد واحدهایش از واحدهای مجاز بیشتر نشود، اینکه پیش نیاز های درس را پاس کرده باشد، اینکه تایم کلاس یا امتحان دو درس با هم تداخل نداشته باشند، و اینکه ظرفیت درس پر نشده باشد.

سپس توابعی برای حذف درس، رفتن به لیست انتظار، بیرون آمدن به لیست انتظار، دیدن درس های این ترم و دیدن وضعیت نهایی ثبت نام هم داریم.

همچنین به مانند قبل برای هر تابع موردنظر، Event مربوطه هم صدا زده می شود که چون نیازی به پیاده سازی توابع نیست دیگر نیاورده ایم. برای مثال هنگام فراخوانی goToWaitingList ، ایونت createWaitingListEvent که در ادامه می بینیم، صدا زده می شود تا تغییرات را نگه داری کنیم.

```
public class Student extends User{
```

```
    private Integer studentNumber;
    private Integer gpa;
    private Integer termUnits;
    private Integer totalUnits;
    private String field;
    private String faculty;

    private HashMap<Integer, HashMap<Offering, Double>> educationalRecord
= new HashMap<>();
    private ArrayList<AddedOffering> offerings;


    public void addToWeeklySchedule(Course course, boolean waiting)
throws ClassesTimeCollisionException,
            ExamsTimeColisionException,
AlreadyAddedCourseToPlanException,
            AlreadyPassedCourseException,
CourseNotPassedPrerequisitesException,
            CourseCapacityException, MaximumAllowedUnitsException {

                //Check if student has already passed the course or not
                //Check if student's units is less than maximum bound
based on gpa
                //Check if student has already passed the prerequisites
courses of this course
                //Check if course's class time doesn't interfere with
other courses
                //Check if course's exam time doesn't interfere with
other courses
                //Check if student has already add the course to his
courses before
                //Check if course has capacity
    }


    public void removeFromWeeklySchedule(String courseCode) throws
CourseNotFoundException {
        //Remove course from schedule if it is in student's courses
    }

    public void goToWaitingList(Offering offering) {
        //Add student to the course waiting list
    }

    public void removeWaitingStatus(Offering offering) {
        //Get out of waiting list of a course
    }
```

```
    public String getTermSchedule() {
        //Get schedule of term for the specified student
    }

    public String getFinalStatusOfEnrollment() {
        //Get status of the registration of the specified user
    }

}
```

## 4)Instructor:

این مورد همان استاد است:

```
import java.util.ArrayList;

public class Instructor extends User{

    private String instructorId;
    private String educationalInfo;
    private String scienceWorks;
    private ArrayList<Offering> allCourses;

}
```

## 5)Course:

این مورد برای اطلاعات درس است:

```
public class Course {

    private String name;
    private int courseCode;
    private int units;
    private LocalDateTime startExam;
    private LocalDateTime endExam;
    private ArrayList<Course> prerequisitesList;

}
```

## 6)Offering:

این مورد برای ارائه های مختلف هر درس است:

```
import java.time.LocalDateTime;
import java.time.LocalTime;
import java.util.ArrayList;

public class Offering extends Course{

   private Integer classCode;
   private Integer termCode;
   private Instructor instructor;
   private LocalTime startTime;
   private LocalTime endTime;
   private ArrayList<Days> classDays;
   private Integer capacity;
   private Integer waitingListCapacity;
   private ArrayList<Student> studentsSignedUpList;
   private ArrayList<Student> studentWaitingList;
   private boolean canBeChosen;

   public enum Days {
       saturday,
       sunday,
       monday,
       tuesday,
       wednesday,
       thursday,
       friday
   }

}
```

## 7)AddedOffering:

این مورد برای هر درس اضافه شده دانشجو است:

```
public class AddedOffering {
   Offering offering;
   Status status = Status.non_finalized;
   boolean wantsToRemove = false;
   boolean isWaiting = false;
```

```
    public AddedOffering(Offering offering, boolean waiting) {
        this.offering = offering;
        this.isWaiting = waiting;
    }

    public enum Status {
        finalized,
        non_finalized
    }

    public void makeFinalize() {
        if (this.status == Status.non_finalized)
            //Add student to studentsSignedUpList

        this.status = Status.finalized;
    }

    public void setToRemove() {
        this.wantsToRemove = true;
    }

    public void cancelRemoving() {
        this.wantsToRemove = false;
    }

    public boolean isWantsToRemove() {
        return wantsToRemove;
    }

    public void changeWaitingToFalse() { this.isWaiting = false; }
}
```

می بینیم اطلاعات گفته شده در بالا، همگی موجود می باشند. همانطور که گفتیم این مورد برای اطلاعات درس اخذ شده توسط دانشجو است.

سپس همانطور که گفتیم، باید یک اینستنس از دیتابیس اصلی بگیریم و اطلاعات را گرفته و در حافظه اصلی ذخیره کنیم. این کار به شکل زیر صورت می گیرد:

## Database:

در این قسمت، علاوه بر اطلاعاتی که می خواهیم ذخیره کنیم، دو تابع برای گرفتن یک نمونه از دیتابیس و گرفتن اطلاعات از APIهای مربوطه داریم.

```
public class Database {
    ArrayList<Student> students;
    ArrayList<Course> courses;
    ArrayList<Offering> offerings;
```

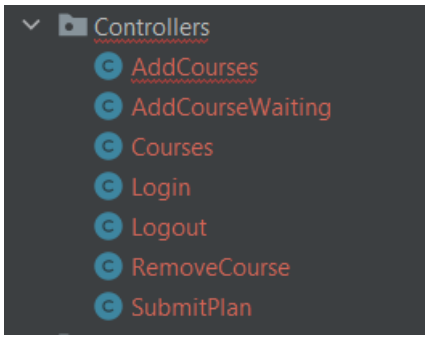
```
    LinkedList<BaseEvent> events = new LinkedList<BaseEvent>();
    Student currentStudent = null;
    private static Database database;

    private Database() {
        students = new ArrayList<Student>();
        courses = new ArrayList<Course>();
        offerings = new ArrayList<Offering>();
    }

    public static Database getInstanceOfDatabase() {
        //Get instance of the main database
    }

    public void getDAtaFromAPI() throws Exception {
        //Get related students, Courses and offerings from main database
    }
}
```

سپس یک سری کنترلر برای هندل کردن درخواست ها داریم. بدین صورت که ابتدا نوع درخواست را تشخیص می دهیم و سپس آن را هندل می کنیم. کنترلر های زیر را برای اعمال مختلف داریم:




برای مثال یکی از آن ها یعنی AddCourse که برای اضافه کردن درس است را ببینیم. این مورد برای هندل کردن درخواست اخذ درس های ارائه شده است که به صورت تک تک به سامانه داده میشود:

## AddCourse Controller:

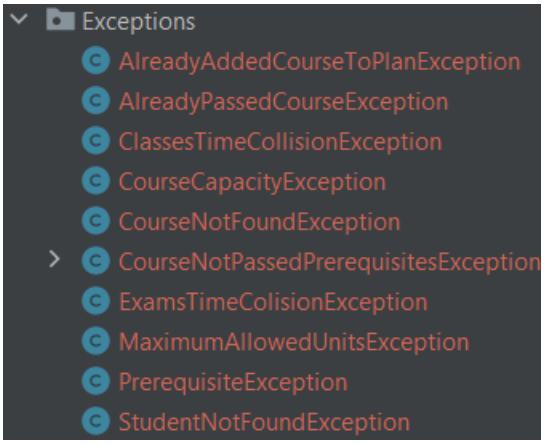
```
public class AddCourses {
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        Database database = Database.getDatabase();
        Student student = database.getCurrentStudent();
```

```
        try {
            student.addToWeeklySchedule(database.getCourse(courseCode,
classCode), false);

        } catch (CourseNotFoundException e) {
            //Send error messages
        } catch (CourseNotPassedPrerequisitesException e) {
            //Send error messages
        } catch (AlreadyPassedCourseException e) {
            //Send error messages
        } catch (ClassesTimeCollisionException e) {
            //Send error messages
        } catch (ExamsTimeColisionException e) {
            //Send error messages
        }
        catch (CourseCapacityException e) {
            //Send error messages
        }
        catch (MaximumAllowedUnitsException e) {
            //Send error messages
        } catch (AlreadyAddedCourseToPlanException e) {
            //Send error messages
        }
    }
}
```

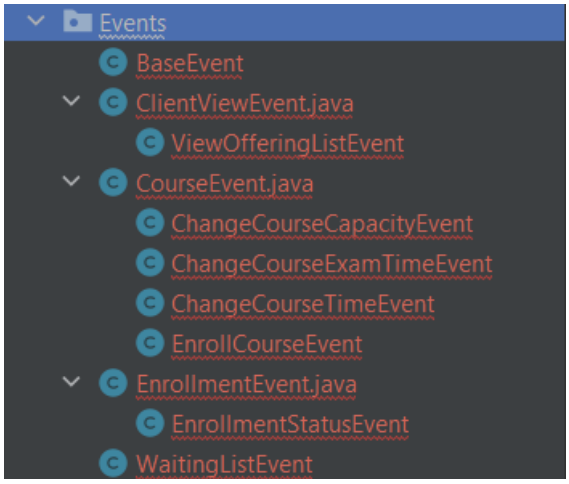
با گرفتن درخواست و تشخیص نوع آن، یک اینستنس از دیتابیس می گیریم و تابع student.addToWeeklySchedule را صدا می زنیم و تمامی قواعد و محدودیت ها هم در این تابع چک می شدند و در صورت بازگشت هر کدام از اکسپشن ها، ارور مربوطه را برمیگردانیم.

همچنین اکسپشن ها هم همگی در فایل Exception موجود می باشند:



در نهایت هم Event ها مختلف برای کامل شدن Event Sourcing را با هم ببینیم.

Eventهای زیر را خواهیم داشت:




در ابتدا یک ایونت پایه داریم که شامل کد ایونت، زمان اتفاق افتادن آن، تایپ آن که چه نوعی است، یوزری که آن را فراخوانی می کند، خواص آن در قالب json، valid بودن یا نبود آن و .... را دارد. سایر event ها از آن ارث بری می کنند.  به عنوان یک توضیح کوتاه به صورت زیر عمل می کنیم:

بر فرض برای اخذ درس جدید، کاربر یک json به ما میفرستد به صورت {course_id:1, student_id:5} که یعنی کاربر ۵ می خواد درس با آیدی یک را بگیرد. ما string همین json را می گیریم و هش این string را به عنوان eventID در نظر میگیریم. بعد در صورتی که کاربر خواست همین درخواست را حذف کند، باید همین json را بفرستد. دوباره hash این string حساب می شود و eventID استخراج می شود. بعد از داخل آن لیست ایونت ها، میتوانیم ایونتی که مربوط به این درخواست بوده را پاک کنیم. بدین ترتیب، برای حذف درخواست باید ایونت منتاظر را پیدا کرد و از داخل آن لیست ایونت ها، حذف کرد.

همچنین تمامی موارد، ایونت جدا ندارند. مثلا برای حذف و اخذ درس دو تا ایونت نداریم، در واقع یک ایونت است که یه پراپرتی eventType دارد. مثلا اگر eventType="/course/enroll" بود، یعنی ایونت اخذ درس است و اگر همان ریکوئست به course/enrollment/remove/ بود، تایپ ریکوئست course/enrollment/remove/ میشود و برای حذف درس می باشد.

توضیحات را به طور خلاصه دادیم. حال کدها را ببینیم:

## 1)BaseEvent:

ایونت ابتدایی برای ارث بری سایر ایونت ها:

```
public class BaseEvent{
    private String eventID; // this could be the hash of request body
    private LocalTime eventTime;
    private String eventType; // could be endpoint eg www.ems.com/course/enroll
    private String eventIssuer;
    private String eventProperties; // properties of event in json string could
be request body
    private boolean isProcessed;
    private boolean isValid;

    public void setEventID() {
        md5 = MessageDigest.getInstance("MD5");
        this.eventID = md5.digest(this.eventProperties);
    }

    public void setEventProperties(String eventProperties) {
        this.eventProperties = eventProperties;
    }
}
```

## 2)ViewOfferingListEvent:

این Event، برای درخواست دیدن لیست ارائه های مختلف درس است. بیشتر برای logging است و شاید بهتر است به آن ایونت نگوییم.

```
// this is not really an event just for logging
public class ViewOfferingListEvent extends BaseEvent{
    public ViewOfferingListEvent() {}
}
```

## 3) EnrollCourseEvent (createEnrollCourseEvent & removeEnrollCourseEvent) :

دومین Event برای زمانی است که یک enrollment به درسی انجام می شود. سومی برای حذف آن enrollment است. این دو ایونت در یک ایونت هستند و جدا نشده اند و همانطور که بالا توضیح دادیم با پراپرتی eventType از هم جدا می شوند:

```
public class EnrollCourseEvent extends BaseEvent{
    private Student student;
    private AddedOffering addedOffering;

    public EnrollCourseEvent createEnrollCourseEvent(String request) {
        //This event handle creating of an enrollment in an specified
course
    }

    public EnrollCourseEvent removeEnrollCourseEvent(String request) {
        //This event handle removing of an enrollment in an specified
course
    }
}
```

### 4) ChangeCourseCapacityEvent:

این ایونت برای تغییر ظرفیت درس است:

```
public class ChangeCourseCapacityEvent extends BaseEvent{
    private Admin admin;
    private Offering offering;
    private int oldCapacity;
    private int newCapacity;

    public ChangeCourseCapacityEvent
createChangeCourseCapacityEvent(String request) {
        //This event handle changing capacity of the specified offering
    }
}
```

### 5 & 6) ChangeCourseTimeEvent & ChangeCourseExamTimeEvent:

این دو ایونت به ترتیب برای تغییر زمان امتحان و زمان کلاس درس هستند.

```
Public class ChangeCourseTimeEvent extends BaseEvent{
    private Admin admin;
    private Offering offering;
```

```
    private LocalTime oldTime;
    private LocalTime newTime;

    public ChangeCourseTimeEvent createChangeCourseTimeEvent(String
request) {
        //This event handle changing class time of the specified offering
    }
}

public class ChangeCourseExamTimeEvent extends BaseEvent{
    private Admin admin;
    private Offering offering;
    private LocalDateTime oldTime;
    private LocalDateTime newTime;

    public ChangeCourseExamTimeEvent
createChangeCourseExamTimeEvent(String request) {
        //This event handle changing exam time of the specified offering
    }
```

### 7) EnrollmentStatusEvent:

این ایونت برای وضعیت درس است که آیا می شود در آن ثبت نام کرد یا خیر:

```
// disable or enable enrollment for student
public class EnrollmentStatusEvent extends BaseEvent{
    private BaseEvent baseEvent;
    private Student student;
    private Admin admin;
    private bool previousState;
    private bool currentState;

    public EnrollmentStatusEvent createEnrollCourseEvent(String request)
{
        //This event handle changing status of an enrollment
    }
}
```

### 8 & 9)WaitingListEvent (createWaitingListEvent & removeWaitingListEvent):

این دو ایونت هم برای اضافه شدن یا حذف شدن در لیست انتظار می باشند.

```java
public class WaitingListEvent extends BaseEvent{
    private BaseEvent baseEvent;
    private Student student;
    private AddedOffering addedOffering;
    private int oldQueueSize;
    private int newQueueSize;

    public WaitingListEvent createWaitingListEvent(String request) {
        //This event handle going to waiting list of a course
    }

    public WaitingListEvent removeWaitingListEvent(String request) {
        //This event handle getting out of waiting list of a course
    }
}
```

در نهایت تمامی مواردی که در این سوال نیاز داشتیم را با هم دیدیم و توضیح دادیم.

# پرسش دوم

با توجه به اینکه از مکانیزم event sourcing استفاده می‌کنیم، به نظر می‌رسد بهتر هست تا هر گونه تغییر در داده‌ها را با استفاده از درخواست‌هایی به سمت سرویس انتخاب واحد انجام دهیم. بنابراین نیاز است تا در سمت سرویس انتخاب واحد تدابیر امنیتی مورد نیاز اعمال شود تا فقط در خواست‌های ارسال شده از طریق سیستم‌های محلی (local) دانشگاه مورد قبول واقع شوند. در واقع از یک فیلتر در اینجا استفاده می‌شود که درخواست‌های مربوط به تغییر داده‌های پایه باید فقط از طریق شبکه داخلی ارسال شده باشند و همچنین علاوه بر آن در صورت نیاز می‌توان از مکانیزم‌های رمزگذاری برای صحت سنجی درخواست دریافت شده نیز استفاده نمود. برای پردازش درخواست‌های انتخاب واحد نیز به مانند قبل رفتار می‌کنیم و تغییری برای آن ایجاد نخواهد شد. همچنین تحت هر شرایطی (حتی زمانی که سامانه زیر load بسیار زیاد قرار دارد نیز) منابع (resource) مورد نیاز برای پاسخگویی به درخواست‌های ارسال شده از سمت شبکه داخلی برای تغییر داده‌های پایه موجود است و اولویت پردازش درخواست‌ها با این درخواست‌ها است.

یکی از مزیت‌های این کار این است که در صورت رخ داد خرابی در سیستم (که در چنین حالتی ممکن است زیاد رخ دهد، با توجه به اینکه ممکن است در ارسال درخواست‌ها خطای انسانی رخ دهد) می‌توان به سادگی سیستم را به حالت پایدار قبلی بازگرداند (در پرسش سوم این مورد توضیح داده شده است).

# پرسش سوم

در اینجا نیاز است تا دو بازه ی زمانی مختلف را بررسی کنیم؛ 1) بازه‌ی انتخاب واحد، 2) زمان های دیگر، اما با توجه به اینکه سیاست در نظر گرفته شده برای این دو بازه تفاوت کمی دارد، بنابراین در ادامه موارد مربوط به بازه‌ی انتخاب واحد را ذکر کرده و هر جا که تفاوتی بود، آن را به صورت صریح بیان خواهیم کرد.

ابتدا به تشریح مکانیزم event sourcing و تصمیمات اتخاذ شده در رابطه با این مکانیزم در سامانه‌ی انتخاب واحد می‌پردازیم.

- با توجه به اینکه در این بازه تمام داده های مورد نیاز بر روی حافظه سامانه انتخاب واحد جمع‌آوری شده‌اند، بنابراین مکانیزم event sourcing را فقط بر روی این سامانه اعمال می‌کنیم.
- از آنجا که اگر بخواهیم تمام event ها را در تمام زمان‌ها ثبت کنیم، بار سیستم به اصطلاح سنگین می‌شود (زیرا هم عملیات ثبت event ها و snapshot گیری سنگین خواهد بود و هم عملیات بازخوانی آنها و همچنین فضای زیادی نیز اشغال می‌شود)، تنها وضعیت اولیه سامانه (مشابه با ذخیره یک snapshot از کل سامانه) و درخواست‌های ارسال شده را به عنوان event ذخیره می‌کنیم.
- همچنین از آنجا که در مکانیزم event sourcing اگر تعداد event ها زیاد باشد، این مکانیزم را با کندی مواجه می‌کند و با توجه به اینکه در این بازه درخواست‌های زیادی به سرویس انتخاب واحد وارد می‌شود (event های بسیار زیادی ثبت می‌شوند)، عملیات به روز رسانی event ها را هر یک ساعت یکبار (در بازه‌ی خارج از انتخاب واحد به صورت روزانه) انجام می‌دهیم. یعنی اگر در یک ساعت گذشته مشکلی وجود نداشته است، دیتابیس مربوط به event های اخیر را به دیتابیس مربوط به کل event ها انتقال داده و سپس داده‌های آن را پاک می‌کنیم تا event های جدید را داخل آن ثبت کنیم.
- در اینجا هدف افزایش سرعت است، بنابراین آخرین event ها و snapshot پایدار را بر روی حافظه اصلی نگهداشته و بقیه موارد (دیگر snapshot ها و event های بیش از یک ساعت گذشته) را بر روی دیسک ذخیره می‌کنیم.
- علاوه بر آن snapshot از آخرین وضعیت پایدار (stable) از سامانه را هم به‌روزرسانی می‌کنیم (یعنی snapshot جدیدی گرفته شده و جایگزین قبلی می‌شود، البته در صورت وجود حافظه کافی می‌توان snapshot قبلی را هم در جایی ذخیره کرد).

بدین صورت هر بار که سیستم با خرابی مواجه شود، یک backup مطمئن وجود خواهد داشت. حال مسئله مهم چگونگی بازگرداندن سامانه به وضعیت «دقیقا» قبل از خرابی است.

با عنایت به این موضوع که بازه‌ی انتخاب واحد یک بازه ی به اصطلاح حساس از سمت دانشگاه است و زمان قطعی سیستم باید تا حد ممکن کمینه باشد، فرض شده است که تمامی موارد زیر به صورت خودکار انجام خواهند شد.

با توجه به اینکه احتمالا یکی از آخرین درخواست‌ها به سمت سامانه باعث ایجاد این خرابی شده است (فرضیه)، به محض تشخیص خرابی در سامانه، به صورت خودکار و موازی چند سرویس دیگر بالا آمده و هر کدام از آخرین snapshot موجود، event ها را به ترتیب رخ داد تا قبل از آخرین درخواست اجرا می‌کنند. در ادامه با یک مثال این روش را توضیح می‌دهیم.

فرض کنید 4 سرویس بالا می‌آیند ، سرویس اول event ها را تا آخرین event ، سرویس دوم event ها را تا یکی به آخرین event ، سرویس سوم event ها را دو تا به آخرین event و سرویس چهارم event ها را تا سه تا به آخرین event اجرا می‌کنند. سرویسی که بتواند event های بیشتری را اجرا کرده و خراب نشود، به عنوان سرویس اصلی بارگذاری می‌شود و event های بعد از آن event مربوط به آن سرویس نیز به عنوان event های مخرب شناسایی شده و در جایی برای بررسی بیشتر ذخیره خواهند شد. اگر همه‌ی 4 سرویس خراب شدند نیز آن 4 سرویس را دوباره راه اندازی کرده و این بار به ترتیب: تا 4 تا مانده به آخرین event ، تا پنج تا مانده به آخرین event ، تا شیش تا مانده به آخرین event و به همین ترتیب event ها را بر روی سامانه‌ها اجرا می‌کنیم تا event مخرب را پیدا کنیم.

همانطور که مشخص است، با توجه به اینکه در این ساز و کار اصل بر پیدا کردن event مخرب است، پس سیستم به دقیقا به وضعیت قبل از خرابی بازخواهد گشت.

تبصره 1: در صورتی که برآورد اولیه در مورد event ها از نظر زمان بازگشت به حالت پایدار اشتباه بوده باشد (یعنی راه اندازی چند سامانه موازی برای تشخیص event مخرب زمان زیادی را بگیرد)، می‌توان بجای اجرای event ها تا رسیدن به event خاصی بر روی سامانه، اجرای event ها را تا رسیدن به زمان خاصی انجام داد. مثلا فقط تا event های مربوط به یک دقیقه قبل از خرابی را اجرا کرد و اگر به حالت پایدار نرسیدیم، تا دو دقیقه قبل از خرابی را اجرا کنیم و به همین صورت ادامه دهیم. در واقع در این روش جدید به دنبال پیدا کردن زمان مخرب هستیم نه event مخرب.

تبصره 2: با توجه به اینکه در این سامانه با External Queries مواجه هستیم (در پرسش دوم توضیح داده شد)، شاید نگرانی‌هایی بابت صحت event ها و درخواست‌های مربوطه موجود باشد، برای جلوگیری از هرگونه عدم

قطعیت در درخواست‌های مربوط به تغییر داده‌های پایه، در event های مربوط به این نوع درخواست‌ها تمام اطلاعات مورد نیاز (علل الخصوص داده‌ی مورد نظر برای تغییر و مقدار جدید آن) ذخیره می‌گردد.

تبصره 3: به منظور سازماندهی مدیریت منطق مربوط به event ها (یا به قول آقای فاولر Structuring the Event Handler Logic) ، از Transaction Scripts استفاده خواهیم کرد. علت این انتخاب نیز این است که همانطور که توضیح داده شد، در اینجا با سیستم ساده‌ای برای این کار مواجه هستیم و بسیاری از دردسرهای مربوط به event sourcing همانند External Updates و غیره در اینجا رخ نخواهند داد. همچنین این یک سامانه داخلی سمت دانشگاه است که تنها با سامانه‌های داخل خود دانشگاه ارتباط دارد که این باعث ساده‌تر شدن کار خواهد شد. علاوه بر این با توجه به اینکه این سامانه بر اساس رخ دادها و دستورات (یا همان درخواست‌های از سمت دانشجویان و کارشناسان آموزش) تغییرات را اعمال می‌کند، استفاده از Transaction Scrpits ساده‌تر و کارآمدتر خواهد بود. ارتباط ساده این سامانه با پایگاه داده و ثبت یا خواندن event ها از آن نیز از دیگر دلایل استفاده از این روش برای مدیریت event ها است.

تبصره 4: در این ساز و کار، عملیات جمع آوری event ها و snapshot گیری به صورت موازی با سامانه اجرا می‌شوند و تداخلی در روند کار سامانه انتخاب واحد ایجاد نمی‌شود.

تبصره 5: درصورتی که خرابی تشخیص داده شد و برای رفع خرابی نیاز به تغییر در کد برنامه بود، پس از رفع باگ مربوطه، سامانه با اطلاعات مربوط به آخرین حالت پایدار از داده‌ها بالا آمده و سپس روند اجرای event ها را (همانطور که در بالا در مورد آن توضیح داده شد) شروع می‌کند.

# پرسش چهارم

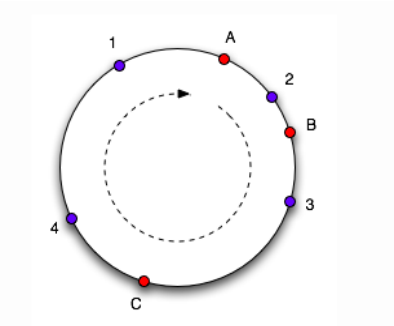
برای انجام توزیع بار در میان سرویس ها (load-balancing) به طور کلی دو روش مختلف وجود دارد link.

روش اولی که به آن اشاره شده مبتنی بر توزیع بار بر روی نسخه های یکسان برنامه است روشی مانند روش تکثیر (Replication) است که چند نمونه یکسان از یک برنامه را اجرا می کنیم و بار را میان آنها توزیع می کنیم.

برای همگام سازی نسخه ها روش های مختلفی وجود دارد روش اول ارسال کپی رخداد ها (event) به دیگر نسخه های برنامه است در این روش باید ابتدا به همه ی دیگر نسخه ها این ایونت را تایید کنند، سپس این ایونت در این نسخه تایید میشود. مشابه روش های رئیس/ دنباله رو (master/slave) به این صورت که دیتا در یک نسخه اصلی تغییر می کند و سپس به بقیه نسخه های کپی ارسال می شود. همچنین برای بهبود شرایط نوشتن و بدون در نظر گرفتن شرط یکنواختی (consistency) در سیستم می توان از روشهای پیچیده تر مبتنی بر رای گیری(quorum) که سعی در کاهش نوشتن در نسخه های برنامه و رای گیری در جواب را دارد استفاده کرد. به این صورت که در نوشتن داده نیاز به تایید همه نسخه ها نیست اما در خواندن نیاز است که از همه ی نسخه ها استیت برنامه را بپرسیم و استیتی با بیشترین رای را به عنوان نسخه درست بپذیریم patterns-of-distributed-systems.

روش دوم روشی است مبتنی بر توزیع بار بر اساس شناسه به شکلی که هر نسخه وضعیت متفاوتی از نسخه دیگر دارد. این روش همانند روش های خرد کردن (sharding) عمل می کند و معمولا بر اساس الگوریتم های هش (hash)عمل کرده و بار را بر اساس یک کلید توزیع میکند.

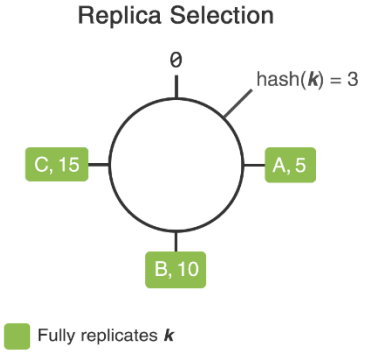
الگوریتم بهینه ی برای اینکار الگوریتم هشینگ پایدار (consistent hashing) است که در صورت تغییر در تعداد نسخه ها توزیع بین نسخه ها مختل نمی شود. consistent hashing




*1-یک نمونه از consistent hashing*

برای استفاده بهینه از این روش بهتر است وابستگی داده ای بین هر نسخه (shard) با دیگر نسخه ها کم باشد تا در صورت نیاز به همگام سازی بار کمتری به سیستم وارد شود. برای مثال برای در صورتی که بر اساس نوع دروس دانشجو (کارشناسی-کارشناسی ارشد-دکتری) تقسیم بندی انجام شود، هر توزیع تا حد زیادی از بقیه نسخه ها متفاوت است. در صورتی که این توزیع مناسب نباشد(تعداد دروس دانشجویان کارشناسی بیشتر از ارشد است و توزیع بار نامتوازن می شود)، می توان بر اساس ترمی که درس باید اخذ شود هم این تقسیم بندی را انجام داد. به این ترتیب بین هر قسمت وابستگی داده کمی است و هزینه همگام سازی کمتر خواهد بود.

برای همگام سازی تا حد ممکن تلاش می شود تا توزیع ها وابستگی کمتری به هم داشته باشند. در غیر اینصورت می توان از روش های گفته شده در قسمت قبل برای این همگام سازی در این روش نیز استفاده کرد. همچنین می توان از روش هایی نظیر تکثیر گذرا (transient replication) استفاده کرد به طوری که نسخه ها بر اساس تابع هش تقسیم بندی می شوند و هر قسمت تکثیر پیدا می کند.




*2-یک نمونه تکثیر گذرا*

روش اول معایبی دارد مانند اینکه همه ی نسخه ها باید تمامی دیتا را در دسترس داشته باشند که در حجم زیاد داده باعث هدر رفتن منابع خیلی زیادی می شود. همچنین در صورت آپدیت یک  داده باید داده در دیگر

نسخه بروز شود که می تواند باعث مشکل غیر یکنواختی (consistency) در سیستم شود و در سیستم حساسی مانند انتخاب واحد نیاز به یکپارچگی کامل تمام نسخه ها هست و باید ظرفیت های دروس به طور دقیق مشخص و بروز رسانی شود و نگهداری و همگام سازی چند نسخه متفاوت از یک داده ممکن است باعث بروز خطا و عدم دریافت داده های درست در بعضی نسخه ها و به طبع آن اختلال در انتخاب واحد شود. همچنین مشکل عدم همزمانی سرویس های (clock synchronization) و همزمان نبودن رخداد ها (event) می تواند باعث بهم ریختگی در ترتیب اجرای رخداد ها شده و باعث خرابی داده ها در بعضی نسخه ها شود. برای رفع این مشکل باید ساعت دقیق هر رخداد به دقت ثبت شود. برای مثال گوگل با استفاده از سامانه های GPS و ساعت های بسیار دقیق اتمی توانسته یک دیتابیس توزیع شده مبتنی بر رخداد (distributed transaction-based database) به نام spanner ایجاد کند.

روش دوم هم معایبی دارد. در صورتی که نسخه های مربوط به یک سری کلید از دسترس خارج شود هیچ نسخه ی دیگری از رخداد ها (event) ها وجود ندارد و عملا دسترسی به سرویس (availability) سیستم دچار اشکال میشود و سامانه برای برخی دروس دچار اختلال می شود. همچنین در صورت عدم توزیع یکسان بار بین توزیع ها ممکن است هر کدام عملکرد متفاوتی داشته باشند.

به طور کلی برای سیستم هایی که میزان همگام سازی داده ها زیاد نیست و بیشتر نیاز به خواندن دیتا هست و نیاز به دسترس پذیری خیلی بالا (availability) و بدون نیاز به یکنواختی (consistency) بالا روش اول بهتر است زیرا به سادگی بار بین نسخه ها توزیع شده و مشکلات مربوط به همگام سازی هم رخ نمی دهد. مانند سیستم های cache و برای سیستم های دیگر روش های مبتنی بر خرد کردن بهتر است زیرا (consistency) را تضمین میکند و منطق برنامه دچار اشکال نمی شود.