Project Description for UniTime Timetabling system: -

By analysing the documentation provided, we find that UniTime it is a scheduling system that allows universities and other organizations to create and manage schedules with ease for both the schedule managers and the students and produce reports for many system aspects, the system can help creating customizable exams & course schedules using automated timetabling algorithms for big organizations like universities and schools without causing conflicts or overlapping according to the courses, the time periods, available rooms and their capacities with regard to preferences and constraints set by the organization allowing for better time management and better sharing of organization resources such as university rooms and instructors.

to have better understanding of the program structure we review classes like TimetableSolver that handles creating timetables for classes using a method like assign that takes a placement as input and check if the placement is valid in the given slots and checks for room and instructor availability and assign the placement if valid and display a message with the reason if not valid and a method as getCBSInfo that takes a classID as input and return statistics about conflicts related to this class

weekSelector class:

 org.unitime.timetable.events;

Before the program's main function, the code creates an instance of WeekSelector widget, the main function determines if a choice has been made for a week between 1 and 52 weeks ago. if not, it starts with week 1 and the choices made for all days will be updated with any selections made for a different week than the one that is currently selected so that they match up.

int firstDayOfWeek = ApplicationProperty.EventGridStartDay.intValue();

The first thing the code does is getting the current day of the week

while (c.get(Calendar.DAY\_OF\_WEEK) != firstDay) {

            c.add(Calendar.DAY\_OF\_YEAR, -1);

        }

This while loop will continue to subtract one day away from the calendar object until it reaches the first day of the week.

while (!c.getTime().after(session.getEventEndDate()))

while loop that determines whether the current time is earlier than the event's end date.

if (c.get(Calendar.YEAR) < sessionYear) {

                Calendar x = Calendar.getInstance(Locale.US);

                x.set(c.get(Calendar.YEAR),11,31,0,0,0);

                dayOfYear -= x.get(Calendar.DAY\_OF\_YEAR);

            }

if clause that determines whether the current year is less than the session year. In that case, it generates a new Calendar object called x and sets the value of it to the last day of the current year. The day of the year of x is then subtracted from the day of the year of c, another Calendar object. This provides the number of days remaining till the year's end.

org.unitime.timetable.events.ListAcademicSessions class:

this class concerns with listing academic sessions either by the current date or by previous date.

We know that by analyzing findSession(org.hibernate.Session hibSession, String term) function.

Its main functionality is taking string it refers to the term of the needed sessions, it retrieves it using Data Access Object pattern (acts like entity manager to separate the data persistence from its logic).

Session ret = SessionDAO.getInstance().get(Long.parseLong(term), hibSession);

If it can’t

It will try to retrieve it from its history by typed queries, then it loops over it to skip the unwanted sessions (empty or test sessions).

List<Session> sessions = hibSession.createQuery("select s from Session s where " +"s.academicTerm || s.academicYear = :term or " +"s.academicTerm || s.academicYear || s.academicInitiative = :term").

        setString("term", term).list();

        if (!sessions.isEmpty()) {

            for (Session session: sessions) {

                if (session.getStatusType() == null || session.getStatusType().isTestSession()) continue;

                return session;

            }

        }

If it cannot, it will check if the entered term is the current term and make another query to get all sessions for today and loops over it to skip the unwanted sessions and retrieve it.

if ("current".equalsIgnoreCase(term)) {

            sessions = hibSession.createQuery("select s from Session s where " +

                    "s.eventBeginDate <= :today and s.eventEndDate >= :today").

                    setDate("today",new Date()).list();

            if (!sessions.isEmpty()) {

                for (Session session: sessions) {

                    if (session.getStatusType() == null || session.getStatusType().isTestSession()) continue;

                    return session;

                }

            }

        }

If it cannot handle the request by get all sessions for the given term, it will throw an exception telling the user that the academic session not found.

throw new GwtRpcException("Academic session " + term + " not found.");

- The project description was obtained using the opportunistic code comprehension technique by firstly applying top-down approach through reviewing the documentation to gain a broad overview of the system and what it does and figure out the key functionalities then we start applying bottom-up approach by getting into the high-level classes and packages that are associated with the main functionalities we recognized about the UniTime system.