CourseNext: Course Management System using Spring Boot

CS 5200 Spring 2018 | Biswaraj Kar | Bishwajeet Dey

Problem Statement

CourseNext is a course management system used by faculty and students. This project is proposed and mentored by Prof. Jose Annunziato. A basic web-based system for the project was made in the past using the MEAN stack. The existing system is fairly basic, web-driven and lacks user/role management, authentication, advanced data-persistence or any kind of relational-schema to interpret and process data. We have been tasked with replacing major parts of the project, as well as enhancing it with new features using Spring Boot and MySQL; to be run on a wholly cloud-based infrastructure with Amazon RDS & AWS Elastic Beanstalk. We will also add advanced role/user management and stateful data-persistence.

CourseNext consists of three types of users or *Person* - *Administrator*, *Faculty* and *Student*. Each *Person* has an account with username and password to configure their profile information. In their profile, they can configure their personal information such as Name, Date of Birth, phones, emails etc. An *Administrator* can add or remove any *Student*, *Faculty* or *Courses* and can access/edit all content on the system. Additionally, they can pull up reports and see overall statistics of the system and its components. The *Faculty* can use the system to upload or create content/course-related material which will be consumed by the students. A *Student* can only view and consume the data setup by the *Faculty*/*Administrator*. A faculty may be part of one or more *Courses* and can create many *Courses*. Each *Course* will have a name, description, create & update dates etc. *Courses* appear on the the home page of the Faculty and may be reordered by the faculty. The layout of the *Course* page will be set using *'Themes'*, where each course will have a theme. The *Theme* will have a name, description, create & update dates and customized styling sheets; which will be set by the Faculty for each course when they create the *Course*. A course may have one or more *Pages*, which link to classroom content for the course. The rendering of a page will depend on its *Layout*, defined using the *Theme* of the course selected earlier. The *Layout* of a page is associated with name, title, description, create & update dates, a background image, and the stylesheet which was set using the Theme. The layout may have many *Pages* which have a name, description, create & update dates. Pages in a course webpage can be reordered by dragging. Each page may have one or more *Tabs* which will act as containers for the actual content and can be arranged in a specific order. Each *Tab* may contain one or more *Widgets*. In the page editor, users can drag and drop widgets from a Widget pallet into the Tabs. *Widgets* can be anywhere on the tabs relative to other widgets and users can drag the widgets to reorder them on the page. There are many different types of widgets, but they all share some common attributes such as an Identifier, name, position (left, right, top, bottom), width, height, foreground & background colors, CSS styling class & sheets etc. A *HTML Widget* will let users add HTML markup. A *GoogleDoc Widget* will let users embed document URLs which are hosted by Google Docs Suite, and these can be Google Documents, Spreadsheets or Presentations. The Google Doc widget additionally lets you set edit/view rights. An *Image Widget* captures an image which has url, horizontal alignment and vertical alignment. Vertical alignment can be top, middle or bottom. Horizontal alignment can be left, center or right. A *Video Widget* lets you embed videos. Each *VideoWidget* is associated with a url and whether it can be expanded, if the Video is an YouTube video, a numeric video Id can be additionally stored.

Proposed Solution

The proposed solution is a cloud-based solution with data stored at Amazon RDS. The following are the expected key features of the project, based on client discussions:

- The system will have advanced User/Role management which will require users to authenticate their credentials first to use any part of the system.
- Based on the authenticated user's role, the User will have varied level of access to the different components of the system. E.g. only Faculty or Administrators will be able to create Courses and Students will only be able to consume the data.
- The data for the application, including all content and layout, will be stored in a relational database (MySQL), which will be optimized for quick queries, data-retrievals and possible analytics in the future.
- The data-persistence infrastructure should be cloud-based to ensure high-availability (Amazon RDS) and 24x7 access.
- There should be some kind of stateful capabilities which will enable faculty to share pages as they are developing it with other faculty with ease.
- The appropriate users (Faculty & Admin) should be able to add Courses, set Themes for them and then add Pages/Tabs/Widgets in an easy-to-use Page Editor Interface.
- There should be proper security implemented at the API level so that only the Users with the correct privileges/roles will be able to access/modify the data.
- The System should be made using Spring Boot application framework for the Java platform. This ensures higher future maintainability and a scalable design as Spring Boot/Java are open-source with a large developer base.

Note that the detailed system-architecture and data-model is yet to be discussed with the client and the above requirements will likely change or enhance over the next one week as we discuss the project with Prof. Jose and finalize the design. The current system is based on the couple of meetings with the client we have had over the past week.

Domain Objects & Enumerations

The key domain objects in the system are:

- **Course, Theme and Layout**: Course is the principal entity of the project and is the primary domain-object of the system. A Course is the highest level entity which represents the grouping of content in the system and is basically the key placeholder and starting point for the Faculty interface. The course has two more 1-to-1 associations, which are to *Theme* and *Layout*. Every course will have only one *Layout*, which is defined by the *Theme* selected for the course.
- **Page**: this is the actual container of the content and represents the logical divisions within a Course. E.g. a Course can be divided into 'Week 1', 'Week 2', 'Week 3' etc. which will be the Pages; or a Web-development course can be divided into Pages named 'Module 1: Basic HTML/CSS', 'Module 2: XML', 'Module 3: AJAX' and so on.
- **Tab**: these are mere arrangements of the data inside a Course for easy organization. E.g. the 'Week 1' page can have 3 tabs: 'Introduction', 'Course Requirements', 'Readings & Projects'. Similarly, the 'Module 1: Basic HTML/CSS' page can have 2 tabs: 'HTML Syntax' and 'CSS Syntax'. The tabs can be reordered as per need and visual convenience.
- **Widget**: this is the most granular entity of the project which represent the course data (Slides, Documents, Presentations, Videos, Images etc.). The tabs will contain one or many Widgets laid out by the Faculty for easy reading and consumption by the Students.
- **Role**, **UserAction, vAlign** and **hAlign**: these are enumerations which defines the specific values which are allowed be set for certain attributes or relationship between Users and domain objects. E.g. *Role* will be used to define the relationship between *Administrator/Faculty/ Student* with the *Course* domain object. On the other hand, *vAlign* or *hAlign* for will be used to set the attribute for vertical or horizontal alignment of the *Image Widget*.

Human Users, their Goals and Relationships

1. **Admin**
   a. Can modify/create/delete/view any User (Faculty/Student) or add new Admins.
   b. Can assign any *Roles* in the system to any User.
   c. Can generate User reports listing all Users.
   d. Can reset passwords for any User in the system and set their Roles/Privileges.
2. **Faculty**
   a. Can create/delete/view any Student.
   b. Can reset passwords for any Student in the system.
   c. Can assign any Role on the Courses owned by them to other Faculty.
3. **Student**
   a. Can view Faculty List (and their courses)
   b. Can reset their own password

User-Domain Relationships

1. **Admin**
    a. Can modify/create/delete/view any Course.
    b. Can modify/create/delete/view/reorder any Page.
    c. Can modify/create/delete/view/reorder any Tab and its Widgets.
    d. Can modify/create/delete/view any *Roles* in the system.
    e. Can see the details of all Courses, Pages, Widgets and can pull up reports/statistics for the same.
    f. Can upload CSS/XML *Themes* to the system for the *Faculty* to use to design their *Course* and its *Pages*.
    g. Can add more *Widgets* to the system to be shown on the Widgets Pallet for Faculty to use them while designing their pages.
2. **Faculty**
    a. Can modify/create/delete/view any Course owned by the Faculty.
    b. Can modify/create/delete/reorder Pages, Tabs and Widgets in any Course owned by them.
    c. Can add custom backgrounds for Themes when setting it for Courses owned by them.
    d. Can see Course report and statistics for the Courses owned by them.
3. **Student**
    a. Can view Courses.
    b. Can view Pages.
    c. Can view Tabs and Widgets.

Domain-Domain Relationships

1. **Course**
    a. A *Course* will be associated with a *Theme* and the corresponding *Layout*
    b. A *Course* may have one or many *Pages*
2. **Page**
    a. A *Page* may contain one or many *Tabs*
    b. The order of the *Pages* in the *Layout* can be changed by the Faculty/Admin.
3. **Tab**
    a. A *Tab* may contain one or many *Widgets*
    b. The order of the *Tabs* in the *Page* can be changed by the Faculty/Admin.
4. **Widget**
    a. A *Widget* can be placed anywhere with respect to other *Widgets* in the *Tab*
    b. An *Image Widget* can have its position defined by *vAlign* or *hAlign* for vertical or horizontal alignment respectively.

Most of the above relationships have been defined in detail and with proper context in the ‘Problem Statement’ section, however, here, we have listed all of them together for clarity.

API Interfaces

1. **Google Apps Script API**: This API allows us to use JavaScript to interact with various Google products that include; Calendar, Docs, Sheets, Slides and more. We plan to use the API for the rendering and editing inside *GoogleDoc Widget.* Faculty will be able to link Documents, Presentations (Google Slides) or Spreadsheets (Google Sheets) from their Google accounts directly into the widget. Also, we plan to implement *Editable* attribute of the Widget using the API, by restricting Read-Only access to some content using this API. The API can also be used to show versions of Documents which can be given as a list for the User to choose from, so that they can choose which version of the document they want to see/show.

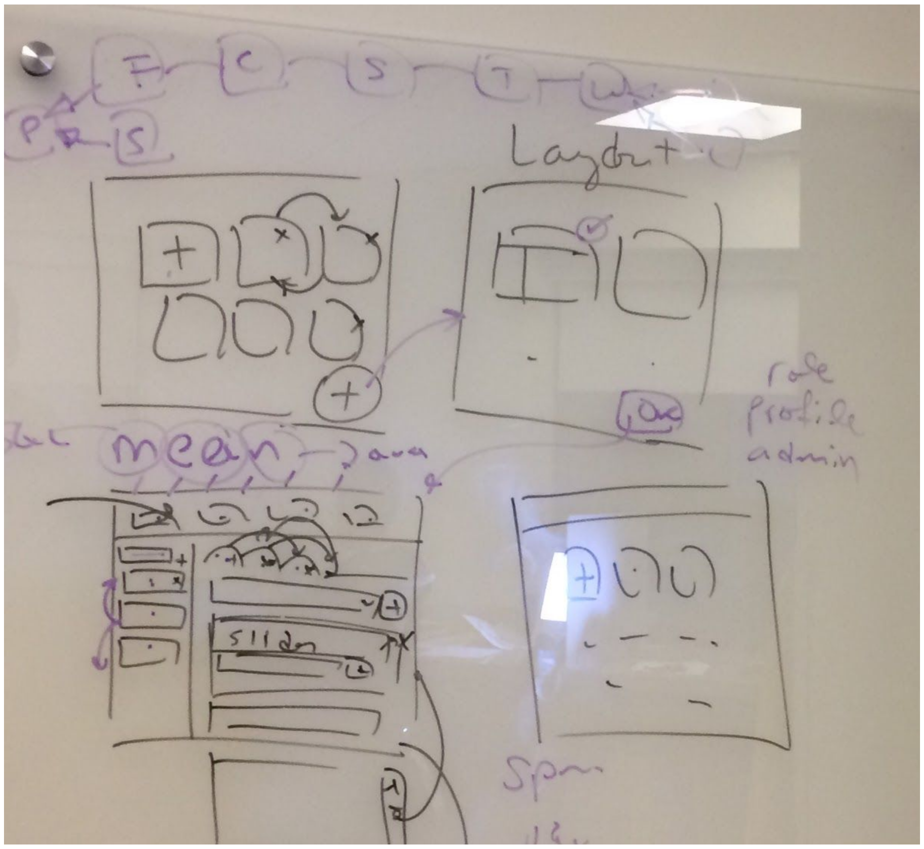
   API Endpoint: https://script.googleapis.com/v1/processes
   API Portal: https://developers.google.com/apps-script/
   API Forum: https://plus.google.com/communities/102471985047225101769
   Developer Support: https://developers.google.com/apps-script/api/support

2. **YouTube API**: This API allows users to integrate their program with YouTube and allow it to perform many of the operations available on the website. It provides the capability to search for videos, retrieve standard feeds, and see related content. A program can also authenticate as a user to upload videos, modify user playlists, and more. We plan to use the API for the rendering and editing inside *Video Widget* when the User uses a YouTube Video or Playlist. Faculty will be able to link their course YouTube Videos or Playlist of Videos directly into the widget. The Widget will allow

   API Endpoint: https://www.youtube.com/iframe_api
   API Portal: https://developers.google.com/youtube/
   API Forum: http://groups.google.com/group/youtube-api/
   Developer Support: http://code.google.com/support/bin/topic.py?topic=12357

Design

The initial design of the project can be described in the form of UML. We have attempted to model the system using Class and Sequence diagrams. Below is the snapshot of the rough system sketch before we dive into the UML diagrams:

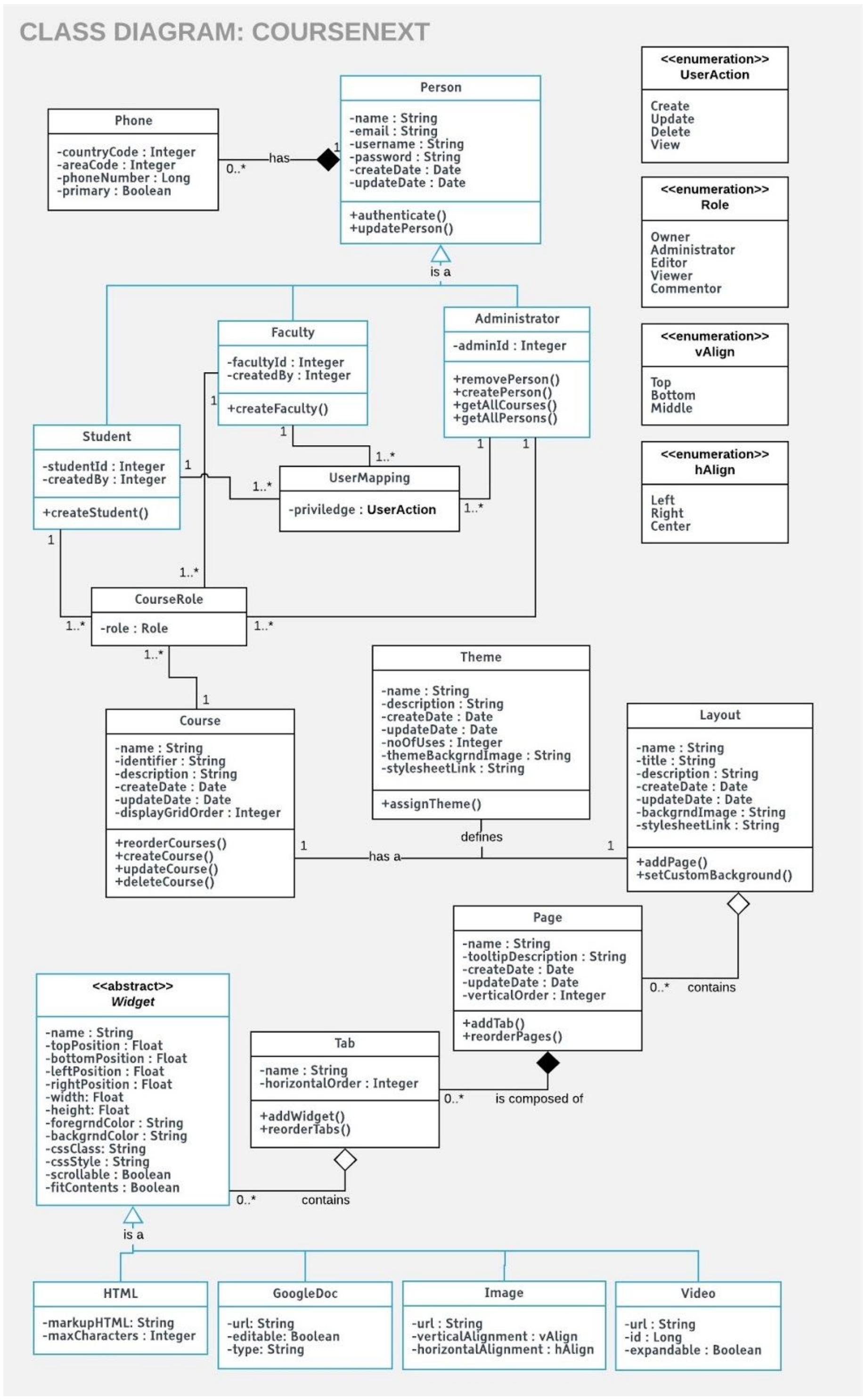


The sketch has 4 UI sketches, as roughly defined by the client.

- The top left sketch represents the landing page for a *Faculty* member, who can add *Courses* and rearrange them on his homepage as per the need.
- The top right sketch represents the UI for selecting a *Theme* for a given course, once the Faculty selects the *Course.*
- The bottom left sketch represents the *Layout* of the *Course* after the *Faculty* has set a *Theme* for the *Course.* The vertical left panel (with small rectangles) are the *Pages* under each *Course* which can be rearranged above/below each other; clicking on which, we get a *Tab* which are shown on the right side of the screen UI. We can see 3 tabs (half semicircular) which can be rearranged. Also, we see a top Horizontal panel, which is nothing but the *Widget Pallet* which has all *Widgets* and they can be dragged onto the tabs.
- Finally, the bottom right sketch shows a simple monolithic page with many options for the *Administrator* (Create User, Create Course, Create Page, Generate Report etc.), each in small square buttons.

UML Class Diagrams:

Below are the snapshots of the UML Class Diagram, the full diagram can be accessed at https://www.lucidchart.com/invitations/accept/0bc92926-bec8-4fda-a904-9ee612f0cd38

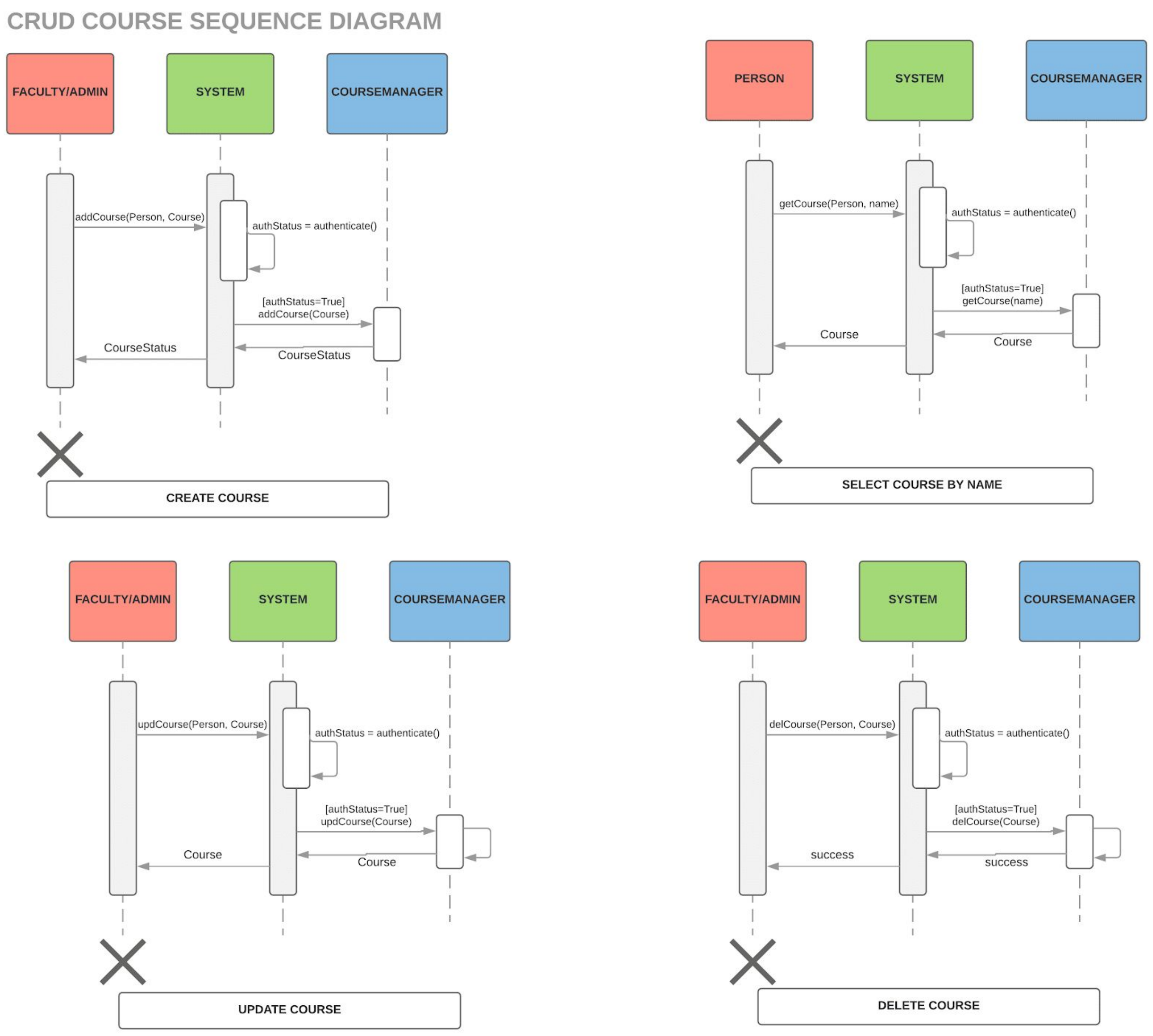


UML Sequence Diagrams:

Below are the snapshots of the UML Sequence Diagrams, the full diagram can be accessed at below URLs:

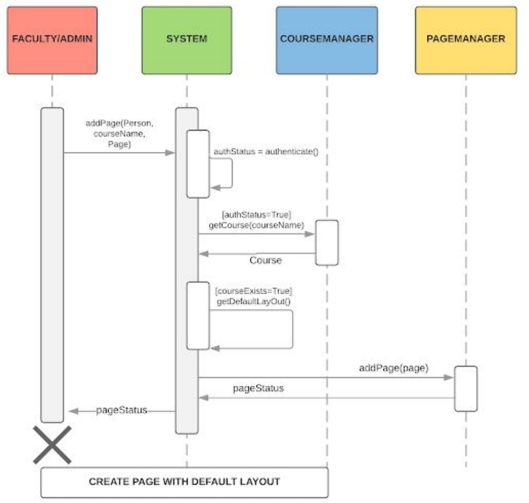
Course:

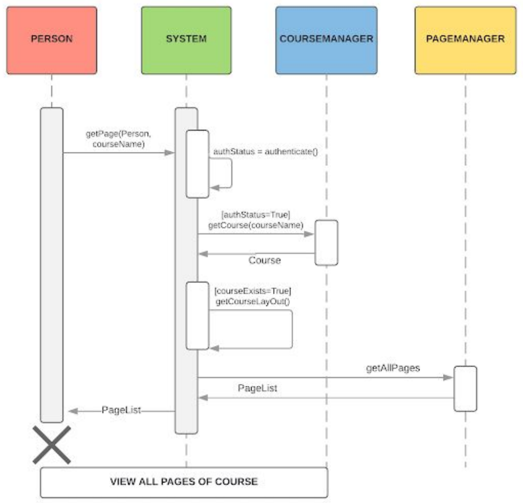
https://www.lucidchart.com/invitations/accept/615e6605-9560-44e1-b745-8178dd30d3e3

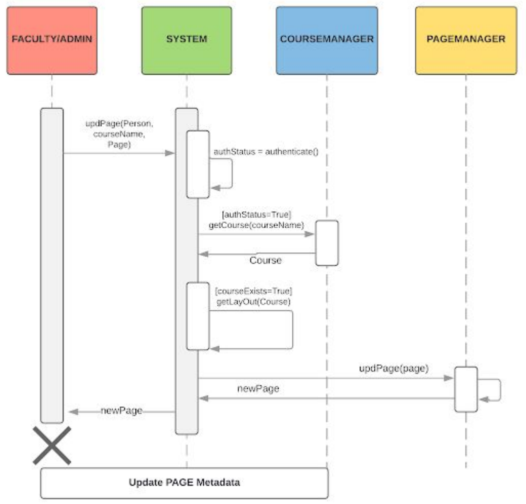

CRUD COURSE SEQUENCE DIAGRAM
FACULTY/ADMIN
SYSTEM
COURSEMANAGER
addCourse(Person, Course)
authStatus = authenticate()
[authStatus=True] addCourse(Course)
CourseStatus
CourseStatus
CREATE COURSE
PERSON
SYSTEM
COURSEMANAGER
getCourse(Person, name)
authStatus = authenticate()
[authStatus=True] getCourse(name)
Course
Course
SELECT COURSE BY NAME
FACULTY/ADMIN
SYSTEM
COURSEMANAGER
updCourse(Person, Course)
authStatus = authenticate()
[authStatus=True] updCourse(Course)
Course
Course
UPDATE COURSE
FACULTY/ADMIN
SYSTEM
COURSEMANAGER
delCourse(Person, Course)
authStatus = authenticate()
[authStatus=True] delCourse(Course)
success
success
DELETE COURSE

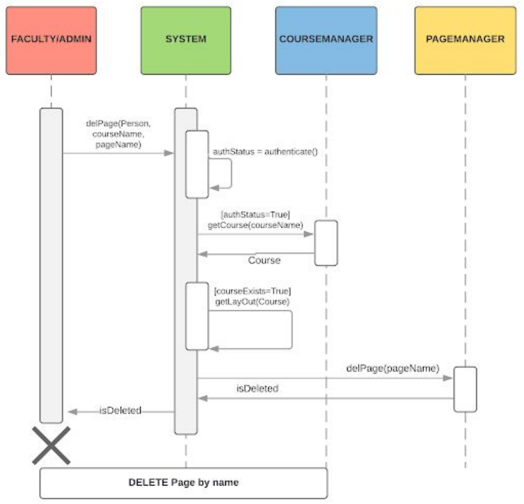
Page: https://www.lucidchart.com/invitations/accept/91fc11ba-ab1f-40a0-a539-03acfa29a67d

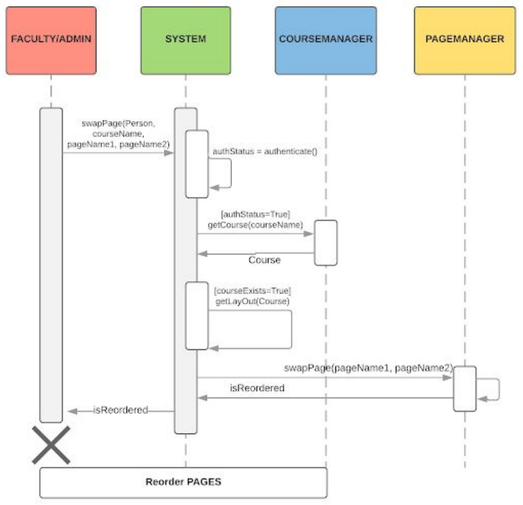
CRUD PAGE SEQUENCE DIAGRAM


FACULTY/ADMIN
SYSTEM
COURSEMANAGER
PAGEMANAGER
addPage(Person, courseName, Page)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getDefaultLayOut()
addPage(page)
pageStatus
pageStatus
CREATE PAGE WITH DEFAULT LAYOUT


PERSON
SYSTEM
COURSEMANAGER
PAGEMANAGER
getPage(Person, courseName)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getCourseLayOut()
getAllPages
PageList
PageList
VIEW ALL PAGES OF COURSE

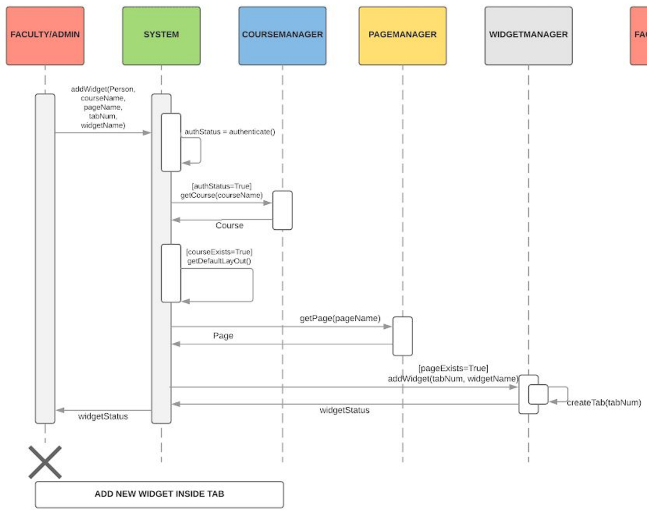

FACULTY/ADMIN
SYSTEM
COURSEMANAGER
PAGEMANAGER
updPage(Person, courseName, Page)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getLayOut(Course)
updPage(page)
newPage
newPage
Update PAGE Metadata

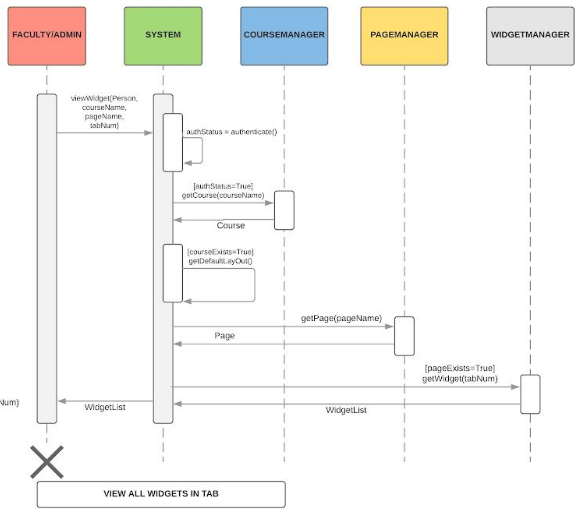

FACULTY/ADMIN
SYSTEM
COURSEMANAGER
PAGEMANAGER
delPage(Person, courseName, pageName)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getLayOut(Course)
delPage(pageName)
isDeleted
isDeleted
DELETE Page by name

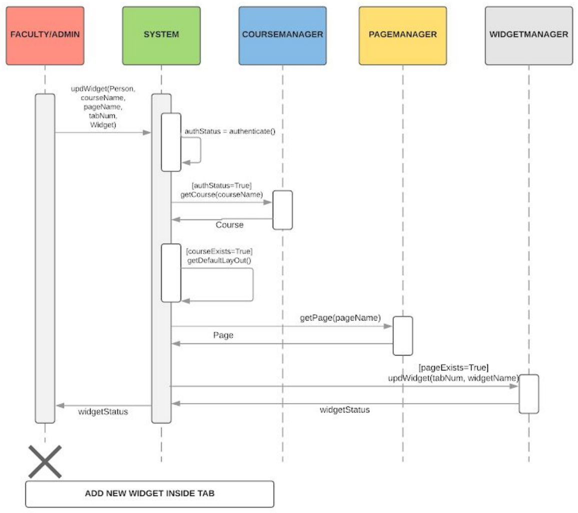

FACULTY/ADMIN
SYSTEM
COURSEMANAGER
PAGEMANAGER
swapPage(Person, courseName, pageName1, pageName2)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getLayOut(Course)
swapPage(pageName1, pageName2)
isReordered
isReordered
Reorder PAGES

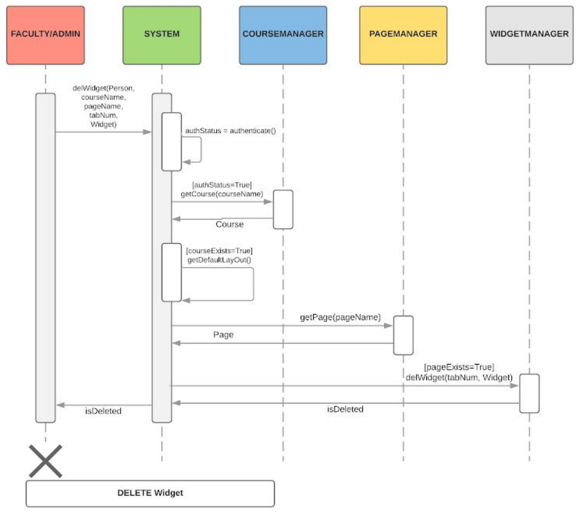
Widget: https://www.lucidchart.com/invitations/accept/acf61c52-0f01-4184-a18a-4e8e769b0bc7

CRUD WIDGET SEQUENCE DIAGRAM


FACULTY/ADMIN
SYSTEM
COURSEMANAGER
PAGEMANAGER
WIDGETMANAGER
addWidget(Person, courseName, pageName, tabNum, widgetName)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getDefaultLayOut()
getPage(pageName)
Page
[pageExists=True] addWidget(tabNum, widgetName)
createTab(tabNum)
widgetStatus
widgetStatus
ADD NEW WIDGET INSIDE TAB


FACULTY/ADMIN
SYSTEM
COURSEMANAGER
PAGEMANAGER
WIDGETMANAGER
viewWidget(Person, courseName, pageName, tabNum)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getDefaultLayOut()
getPage(pageName)
Page
[pageExists=True] getWidget(tabNum)
WidgetList
WidgetList
VIEW ALL WIDGETS IN TAB


FACULTY/ADMIN
SYSTEM
COURSEMANAGER
PAGEMANAGER
WIDGETMANAGER
updWidget(Person, courseName, pageName, tabNum, Widget)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getDefaultLayOut()
getPage(pageName)
Page
[pageExists=True] updWidget(tabNum, widgetName)
widgetStatus
widgetStatus
ADD NEW WIDGET INSIDE TAB


FACULTY/ADMIN
SYSTEM
COURSEMANAGER
PAGEMANAGER
WIDGETMANAGER
delWidget(Person, courseName, pageName, tabNum, Widget)
authStatus = authenticate()
[authStatus=True] getCourse(courseName)
Course
[courseExists=True] getDefaultLayOut()
getPage(pageName)
Page
[pageExists=True] delWidget(tabNum, Widget)
isDeleted
isDeleted
DELETE Widget