# Enhancement of Bodhitree through the Development of Additional Features

Avijeet Gaikwad
143050101

Guided by
Prof. Kameswari Chebrolu

Department of Computer Science and Engineering
IIT Bombay

27th June, 2016
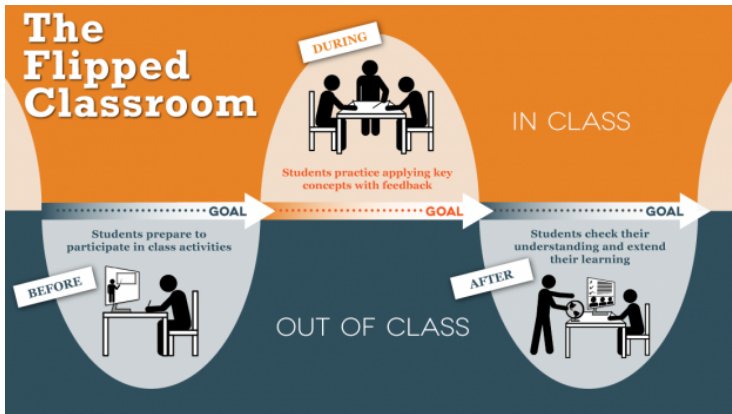
# Outline

# Outline

- Bodhitree is an e-learning platform
  - Design to mimic classroom setting (Bodhi-class)
  - Host multimedia textbooks (Bodhi-book)
- Types of users:
  - Content developer - *Content creation*
  - Instructor - *Course offering*
  - Student - *Register for the course offering*

Source:[1] The University of Texas at Austin: Faculty Innovation Center

Figure : Flipped classroom model

# Bodhi-book
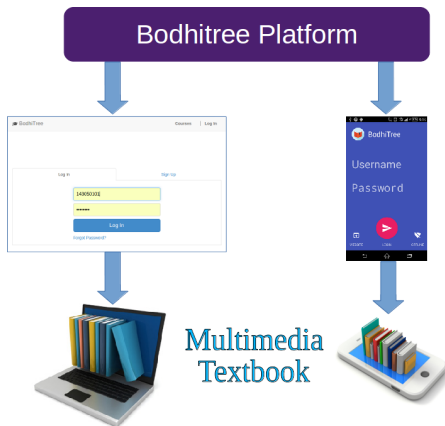A multimedia textbook model
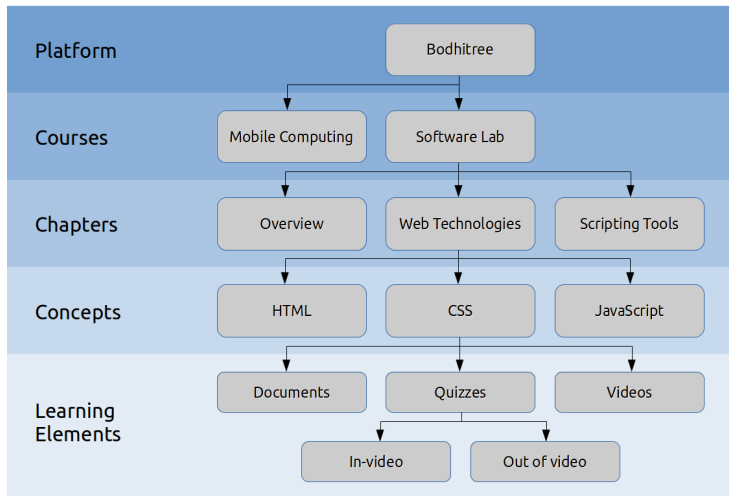


Figure : Multimedia textbook model

# Overview of the components that constitute Bodhitree

- Courseware
    - Chapters
    - Concepts
    - Progress
- Learning elements
    - Videos
    - Quiz
        - In-video quiz
        - Out of video quiz
    - Documents
- Interaction
    - Discussion forums
    - Chat
- Labs
    - Report grading
    - Assignments

## Components of Bodhitree (Hierarchy)

# Development details

- Django web framework (Back-end)
    - (M) Designing the model
    - (V) Creating the templates
    - (C) Writing the views and mapping the URLs

- React.js (Front-end)
    - Component based design
    - Virtual DOM

# Outline

# Objective and Motivation

- Objective is to enhance the Bodhitree platform through the addition of new features
- Need for new features, development ideas:
    - Instructor requirements
    - Student's and Instructor's feedback
    - Brainstorming sessions
- Development of features motivated by:
    - Bodhi-class
    - Bodhi-book

# Overview of the work

- Bodhi-class
  - Marks for offline exams (Stage I)
  - Archiving emails
  - Facebook style notifications
  - Miscellaneous
    - In-video quizzes ON/OFF
    - Setting importance to threads
    - Email instructors when a new thread is added
- Bodhi-book
  - Prerequisites and course graph
  - Access control (Stage I, Modified in Stage II)

# Outline

# Outline

# Marks for offline examinations

## Problem Statement

- Several offline exams are conducted in the Bodhi-class model
- The marks for these offline exams are usually sent by emails, having CSV files attachments
- It is required to have an interface to display these marks to the students on Bodhitree itself

**Instructor Specifications**

- Upload a CSV containing the students marks
- The file should have the exam details and marks, along with the usernames of the students, in the following format

**Students Specifications**

- Students must be able to view only their marks that are uploaded for the course

**User Interface as seen by the instructor**

| Upload Marks | |
|---|---|
| **Marks File (csv)** | Choose File   No file chosen |
| | Upload |

| Student | Q1 (20) | Midsem (30) | Q2 (20) | Endsem (30) |
|---|---|---|---|---|
| root | 18 | 25 | 12 | 22 |
| avi | 12 | 20 | 14 | 24 |

Figure : Instructor's view of CSV upload and display of students marks

**User Interface as seen by the student**

| Q1 (20) | Midsem (30) | Q2 (20) | Endsem (30) |
|---------|-------------|---------|-------------|
| 12      | 20          | 14      | 24          |

Figure : Student 2's view of his marks

**Uploading CSV file**

| EXAM_TYPE | Q1 | Midsem | Q2 | Endsem |
|-----------|----|--------|----|--------|
| **MAX_MARKS** | 20 | 30 | 20 | 30 |
| Student 1 | 18 | 25 | 12 | 22 |
| Student 2 | 12 | 20 | 14 | 24 |

**Storing marks in the database**

| id | course_id | exam_type | max_marks |
|----|-----------|-----------|-----------|
| 1  | 2         | Q1        | 20        |
| 2  | 2         | Midsem    | 30        |
| 3  | 2         | Q2        | 20        |
| 4  | 2         | Endsem    | 30        |

| id | marksheader_id | student_id | marks |
|----|----------------|------------|-------|
| 1  | 1              | 1          | 18    |
| 2  | 2              | 1          | 25    |
| 3  | 3              | 1          | 12    |
| 4  | 4              | 1          | 22    |
| 5  | 1              | 10         | 12    |
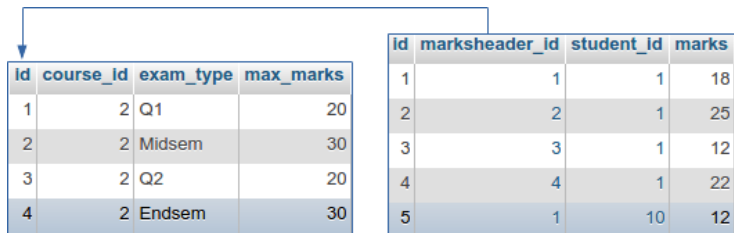
Exams details          Marks obtained by students

Figure : Data view of the marks module

# Outline

# Archiving of emails

## Problem Statement

- Instructors frequently send emails in the Bodhi-class
- These emails are concerned with important announcements and updates in the course
- No easy way for the instructor to view the emails
- Need for archiving and course-wise categorization

# Specifications

- All the emails sent by the instructors must be archived
- Instructor should be able to view all the emails sent by him in a particular course
- Student must be able to view all the emails that are received by him
- Sorting and searching features should be available

**Archiving the emails**

- Function *archive_mail()* called whenever an email is sent
- Foreign keys to the current *user*, *course* and *list of recipients* (many to many relation)
- Postgres automatically adds the current date and time

**Displaying the archived emails**

- User makes an AJAX request to the server
- Server checks the dtails of the user in the current session
- Data is fetched in JSON format

```
GET /email_archive/api/getEmails/1/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "emails": [
        {
            "body": "<p>test</p>",
            "recipients_list": "['avijeet7@gmail.com', 'avijeet@cse.iitb.ac.in']",
            "sender_id": 1,
            "cc_list": "[]",
            "from_email": "instructor_CS224:_Computer_Networks@bodhitree.cse.iitb.ac.in",
            "date_sent": "2016-06-14T15:19:06.564Z",
            "reply_to": "avijeetjob@gmail.com",
            "id": 1,
            "course_id": 1,
            "subject": "test"
        }
    ],
    "mode": "S"
}
```

**UI for viewing the emails**

- Component based design using React.js
- Sorted by most recent first

## Search and filter functions

- Searchbox and date filter
- *Instant search*

# Outline

# Facebook style notifications

## Problem Statement

- Frequent updates in courseware
- Students may be unaware of the updates
- Need for an interface for quick navigation to the updates

# Specifications

- Notifications must cover the following updates in a course:
    - Publishing a new concept
    - Addition of new videos and documents
    - Publishing new quizzes
    - Uploading marksheet
    - Addition of new threads in discussion forum
- Pending notification must be displayed
- Hyperlinks must be provided for easy navigation

**Storing the notifications (PUSH)**

1. **Notice** table stores the notification data
   - *course*: Foreign key to the course id
   - *notice_data*: Data displayed in the notification
   - *notice_url*: Hyperlinks of the updates
   - *notice_date*: Date and time

2. **ClearNotice** table stores the notifications clearing time
   - *user*: One to one mapping to the current user
   - *clear_time*: Stores time of clearing the notification

A function **add_new_notice()** has been created which pushes the notifications to the database

# Design
## Retrieving the notifications
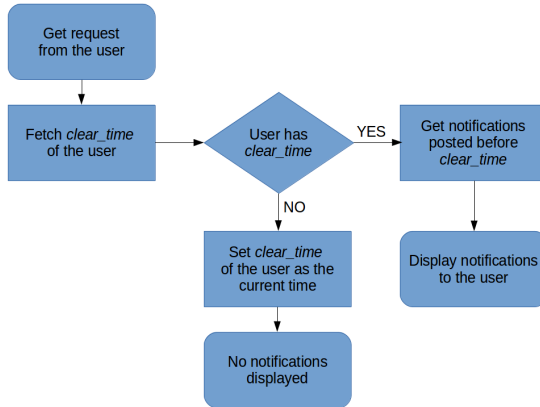


Figure : Flow chart showing the working of notifications retrieval

**User Interface**

- A badge shows the number of pending notifications
- Drop down list shows the categorized view
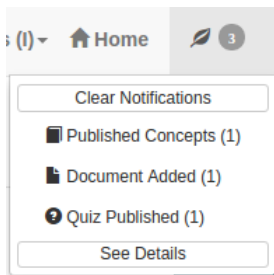


Figure : Notifications indicator along with the drop down list

## Detailed notifications view

- Every single notifications is shown in a categorized way
- These notifications can be used for navigation to those updates

**Clearing the notifications**

- The notifications that have been added after the user has cleared the notifications are fetched from the database
- On clearing, the current time is added against the user's name in the *clear_time* field of the **ClearNotice** database table

# Outline

**Some of the minor additions to Bodhitree:**

- In-Video Quiz ON/OFF
- Setting importance to threads
- Email instructors on addition of a new thread

## Problem Statement

- Several videos have in-video quizzes
- Quizzes pop up and interrupt the play
- Choice for an uninterrupted playback is desirable

# Implementation
User Interface

- Button created using CSS and JavaScript to toggle the video markers
- Navigation links in TOC still present, only the pop-ups are disabled
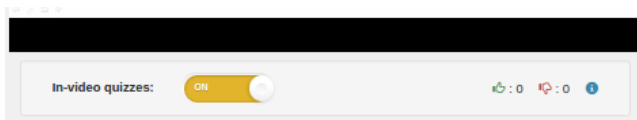


Figure : A button to turn the in-video quiz pop-up ON/OFF

## Implementation
### Mechanism used

- The video container is a component of React.js which controls the display and playback of the video
- Video markers are passed as properties to the video container
- When the video container loads, the value of the button is checked
- ON, in-video quiz data passed to the video container
- OFF, data not passed, container reloads the video without the in-video quiz data

# Setting importance to threads

## Problem Statement

- Threads in the discussion forum can be sorted based on: *'popularity'* and *'earlier first'*

- Some threads are more important than the others, option to set importance value and sort based on *importance* must be present

- When a new thread is added, the default importance value is 0
- Instructor is able to set the value to a number between 0 and 10
- User viewing the discussion forum are able to sort the threads according to the importance set by the instructor

# Email instructors when a new thread is added

## Problem Statement

- Some threads in the discussion forum are started by the students seeking immediate help in certain aspects of the course
- The instructor will be aware of the thread only after checking the discussion forums.
- It is important that the instructor must be notified when new threads are started in the discussion forum

# Implementation

- Whenever a new thread is started in a course, all the instructors of the course are fetched (course creator as well as the TAs)
- An email is sent to all of them, a sample is shown below
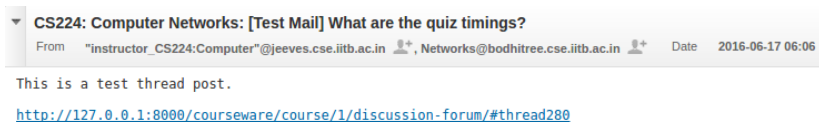


CS224: Computer Networks: [Test Mail] What are the quiz timings?
From "instructor_CS224:Computer"@jeeves.cse.iitb.ac.in , Networks@bodhitree.cse.iitb.ac.in    Date    2016-06-17 06:06

This is a test thread post.

http://127.0.0.1:8000/courseware/course/1/discussion-forum/#thread280

# Outline

# Outline

# Prerequisites and course graph

## Problem Statement

- Bodhi-book model lacks significant involvement of instructors
- Dependencies in chapters and concepts
- Proficiency levels required for complete understanding
- Guidelines are necessary for students
- Graphical representation of the course and the guidelines help the students to progress in a better way

**Motivation is to ensure that every student has the required background knowledge before attempting to learn a new concept**

- Instructor specifies prerequisites for a concept
- Prerequisites viewable on the concept page
- Graph generated using the course data, which serves as a guideline
- Instructor modifies and finalizes the course graph

**Storing the prerequisites**

- Concepts/Chapters are the prerequisites for other Concepts/Chapters
- Prerequisites are stored as a list of Concept/Chapter id's
- Default values are a blank list *[ ]*
- In case of a Chapter, a *"g"* character is appended in front of the id
- Example of prerequisites data for a concept:
  *["g1", "5", "6"]*

# Design
## UI to add prerequisites

- Only the instructor has authority
- Concepts categorized into chapters
- Component based design, immediate request
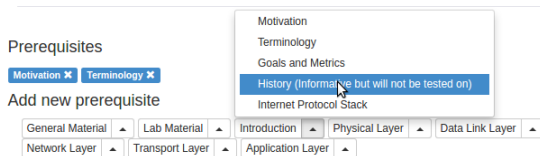


Figure : Adding prerequisite to a concept

**Student's view of prerequisites on the concept page**

**Prerequisites**

Motivation  Terminology  History (Informative but will not be tested on)

- Navigable links

Several graph generation libraries used to generate initial course graphs:

- **Graphviz:** Python library which generates an image of the final graph, non-interactive
- **Arbor.js:** Javascript library which dynamically generates interactive graphs, allows users to interact with the components
- **Vis.js:** Javascript library, interactive graphs, GUI to create/modify existing graphs

**Graph generated using Graphviz library**
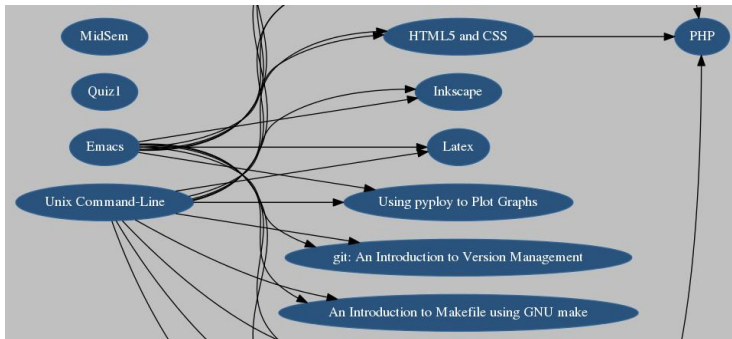


Figure : Part of a complex graph generated using graphviz

**Graph generated using Arbor.js**
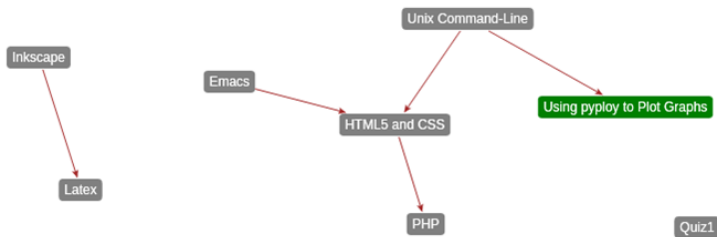


Figure : Part of a graph generated using arbor.js

**Graph generated using Vis.js**
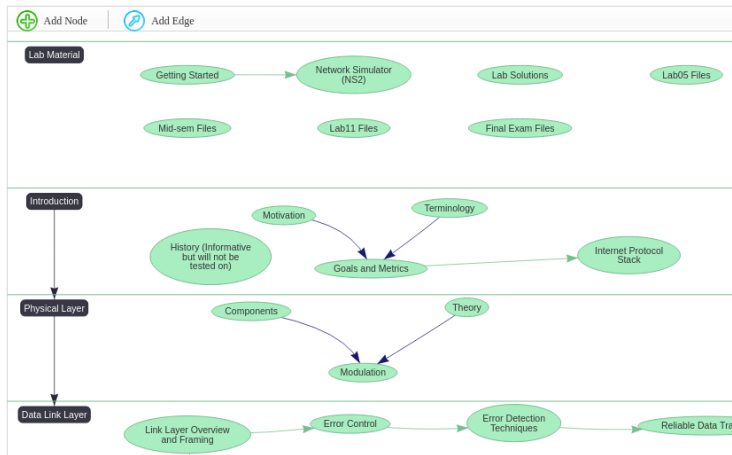
- Function created to fetch the course data, convert it to JSON and render the graph (html file)
- JSON captured on the client side. Function for creation and organization of nodes and edges written in JavaScript
- Instructor must save the graph before the student can view it
- All the components on the canvas are interactive and modifiable
- Example of Nodes and Edges data:

| Nodes | Edges |
|---|---|
| [{id: 1, label: 'Node 1'}, | [{from: 1, to: 3}, |
| {id: 2, label: 'Node 2'}, | {from: 1, to: 2}, |
| {id: 3, label: 'Node 3'}, | {from: 2, to: 4}, |
| {id: 4, label: 'Node 4'}, | {from: 2, to: 5}] |
| {id: 5, label: 'Node 5'}] | |

# Future work
Ideas to empower course graphs

- Using data from the progress module to display proficiency levels of students
- Generating dynamic, student specific graphs for providing alternative learning paths
- Creating multidimensional graphs, which display in-depth information and provide navigation to courseware elements

# Outline

# Access control
Motivation

**Providing differential access to users towards the content**
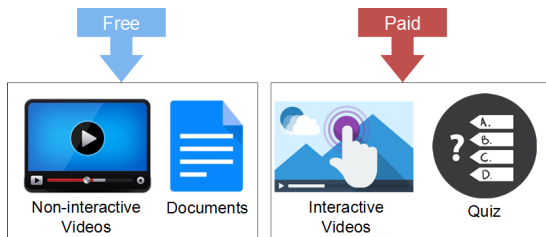


## Problem Statement

- Giving the instructor authority to limit the access a student has to the content in his course

**8 doors, 4 users, 1 owner, 4 types of locks and their keys**
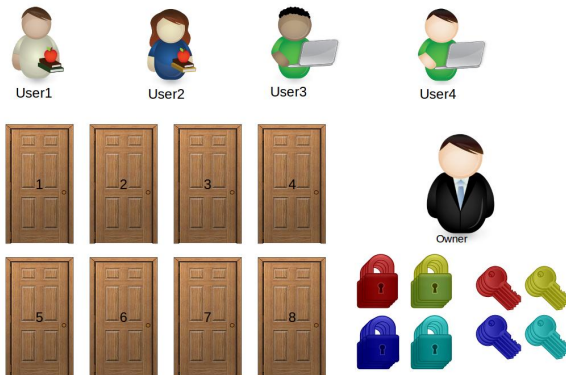
**Owner locks door no. 8, thus restricting access to it**
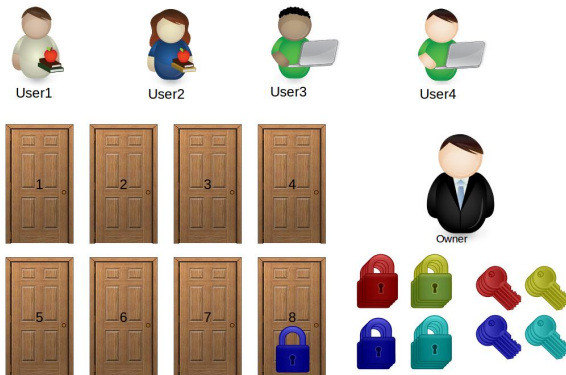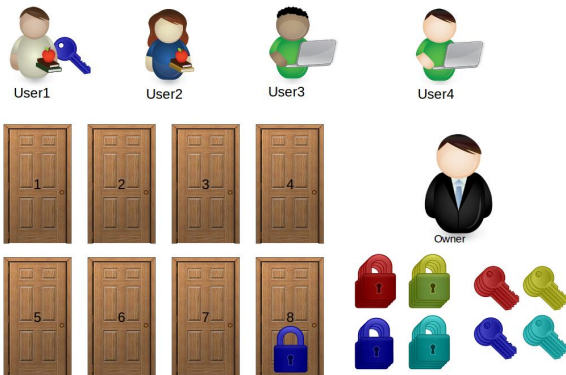
**User1 gets access to door 8**

**Giving access to other users**
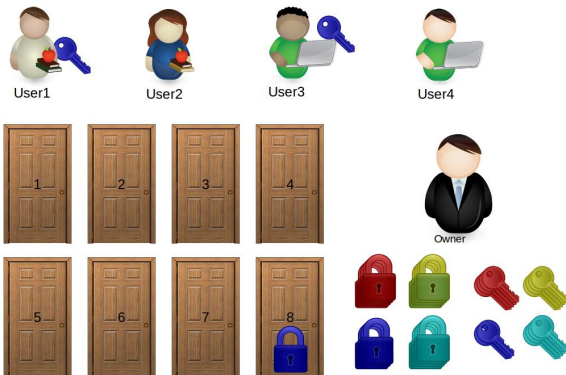
**A different type of lock**
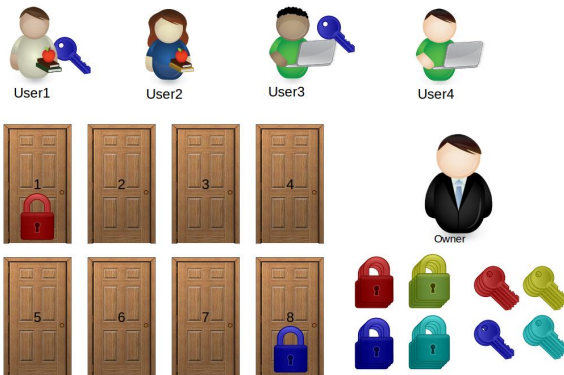
**User1 gets access to multiple doors**
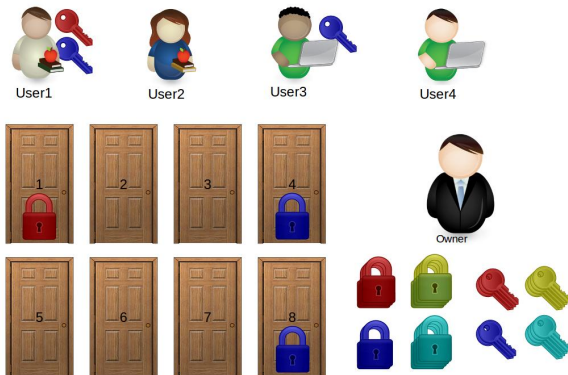
**Restricting access by taking away the key**

# Specifications
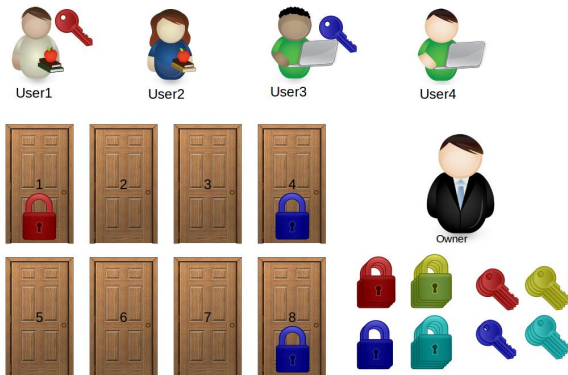
- Analogy
    - Owner: Instructor
    - Users: Students.
    - Doors/rooms: Content offered on Bodhitree.
    - Locks: Tagging of content by an instructor.
    - Keys: Granting of access to the students for those tags.
- Learning elements that can be tagged:
    - Videos
    - Documents
    - Quizzes
    - Video markers (In-video Quizzes)

# Design
## Defaults

- When an element (video, document, quiz or video markers) is created in a concept, the default tag is set as *"Free"*
- A *"Free"* tag denotes that all the students who are registered for that course can access that element
- Fields *premium_tag* and *premium_marker* added in the database

Figure : Tagging interface

- A badge shows the currently set tag
- A drop down box which lists the 5 fixed tags (P1, P2, P3, P4, P5), and the custom tags
- Setting a new tag changes the *premium_tag* field in the database

**Student's access to a learning element**

## Design
Student view
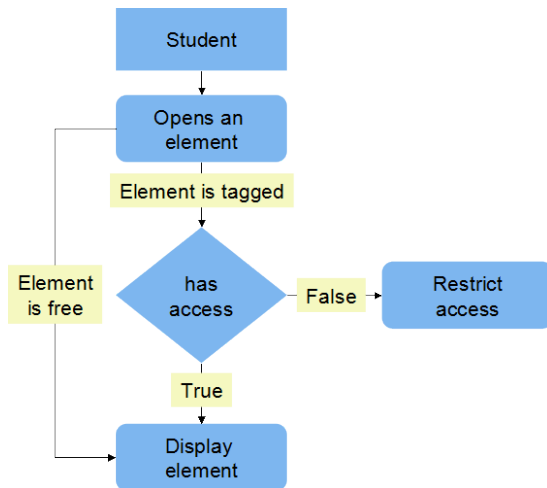
- Server check the user request and sends appropriate data
- Data is not sent to unauthorized users
- The field *"has_access"* is set to *false* in the JSON that is fetched, resulting in the following message

You do not have access to this tag!

P1

Request access
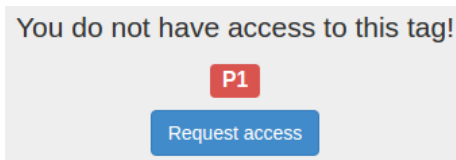
**Data sent to client who is not having access to an element**

```
"content": {
    "id": 6,
    "title": "Components",
    "content": "The video will cover the details of the components that make up the PHY layer.",
    "upvotes": 0,
    "downvotes": 0,
    "video_file": "",
    "markers": [],
    "other_file": "",
    "duration": 0,
    "premium_tag": "P1",
    "premium_marker": "M1"
},
"marker_access": false,
"has_access": false,
"type": "video",
"history": {
```

- Uploading the CSV file:

| stud1 | P1 | P2 | P3 |
|-------|----|----|----|
| stud2 | P1 |    |    |
| stud3 | P3 |    |    |

- Checking username against the registered users
- Storing the tags in the database:

| id | user_id | course_id | premium_type |
|----|---------|-----------|--------------|
| 10 | 1 | 1 | P1 |
| 5 | 1 | 1 | P2 |
| 8 | 1 | 1 | P3 |
| 4 | 1 | 3 | P1 |

Interface to upload the CSV file:

Information displayed after successfully uploading a CSV file

**Removing of tags access from users**

- Content developer has the authority to remove the tags access
- Checklist provided listing the users and associated tags
- Corresponding entries removed from the database

## Interface to remove tag access



List of users who have access to premium features:

| Username | Access Type |
|----------|-------------|
| ☐ 👤 root | P2 |
| ☐ 👤 avi | P1 |
| ☐ 👤 salman | P2 |

Delete

**Creation of custom tags by the instructor**

- If the 5 default tags **(P1, P2, P3, P4, P5)** are not enough, the instructor may create new tags of his own
- Once created, custom tags ar available in the drop down list of the tagging interface

**Interface for adding custom tags**

**Custom tags available in the tagging interface after creation**

**Tag all the elements (Videos, Documents, etc.) in one click**

# Outline

- All the features that are developed as a part of this project are covered in the report and in this presentation
- A lot of time is invested in testing the features and ensuring that they work correctly
- Unit test are written covering every line of the code, the aim being to ensure that the changes that future changes do not break these features
- Complete code was written in accordance to the PEP 8 standards
- The code that was pushed to Bodhitree as a part of this work was peer reviewed either by **Prasanth P** or by **Dheerendra Rathor**

# References I

📄 The University of Texas at Austin: Faculty Innovation Center
https://facultyinnovate.utexas.edu/teaching/flipping-a-class

📄 Bodhitree website
http://bodhitree.cse.iitb.ac.in/mission/

📄 Django, The web framework for perfectionists with deadlines
https://docs.djangoproject.com/en/1.9/

📄 React, A javascript library for building user interfaces
https://facebook.github.io/react/docs/why-react.html