
Graduation Projects Recommendations Website
Supervised by
Dr. Moustafa Ezzat

Problem

Computer science students can easily obtain Graduation projects ideas with help of many websites, which are interested in the fields of software development.

Graduation projects ideas don't satisfy the tools, that the students can work with-it.

So, we develop a system to recommend graduation projects based on the tools, which students can work with-it.

Scope

- **Our system is based on “Stack overflow” survey, which investigate the most active application ideas and the tools are used in them.**
- **The system receives the tools mastered by the students.**
- **Then the system recommend 5 projects ideas, that fit the mastered tools.**
- **The system provides a brief for each recommended project.**
- **The system supports team-work concept.**

Stakeholder

Stakeholder is a beneficiary who wants to take advantage of our system.

Final-year students are the main stakeholder in our project.

The system saves students' efforts, of searching for graduation projects without a reference.

Development Methodology

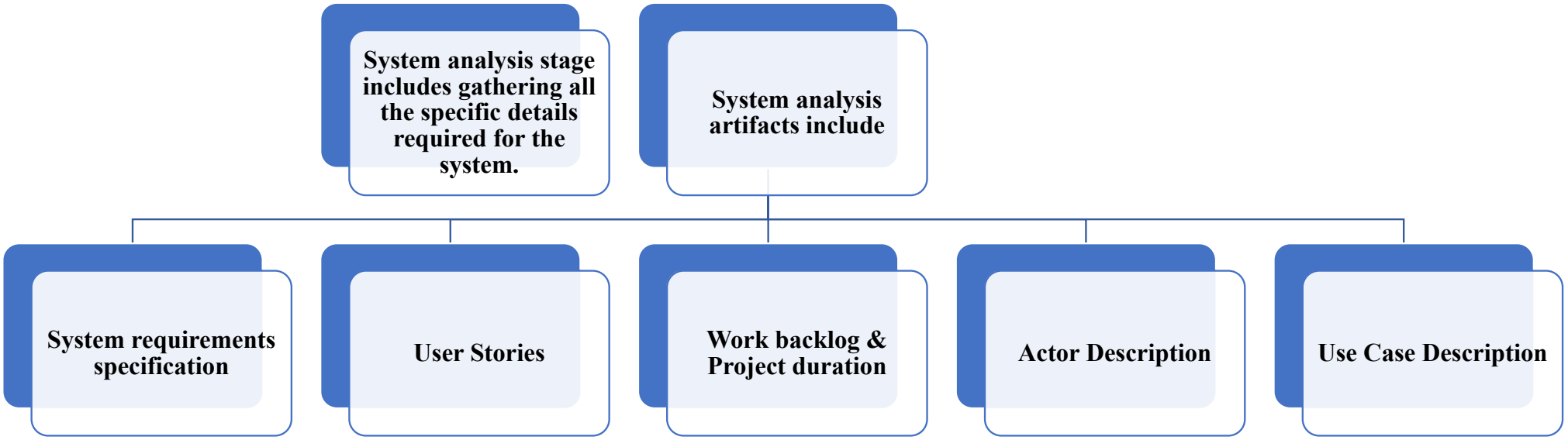
Scrum Agile methodology is our system development methodology.

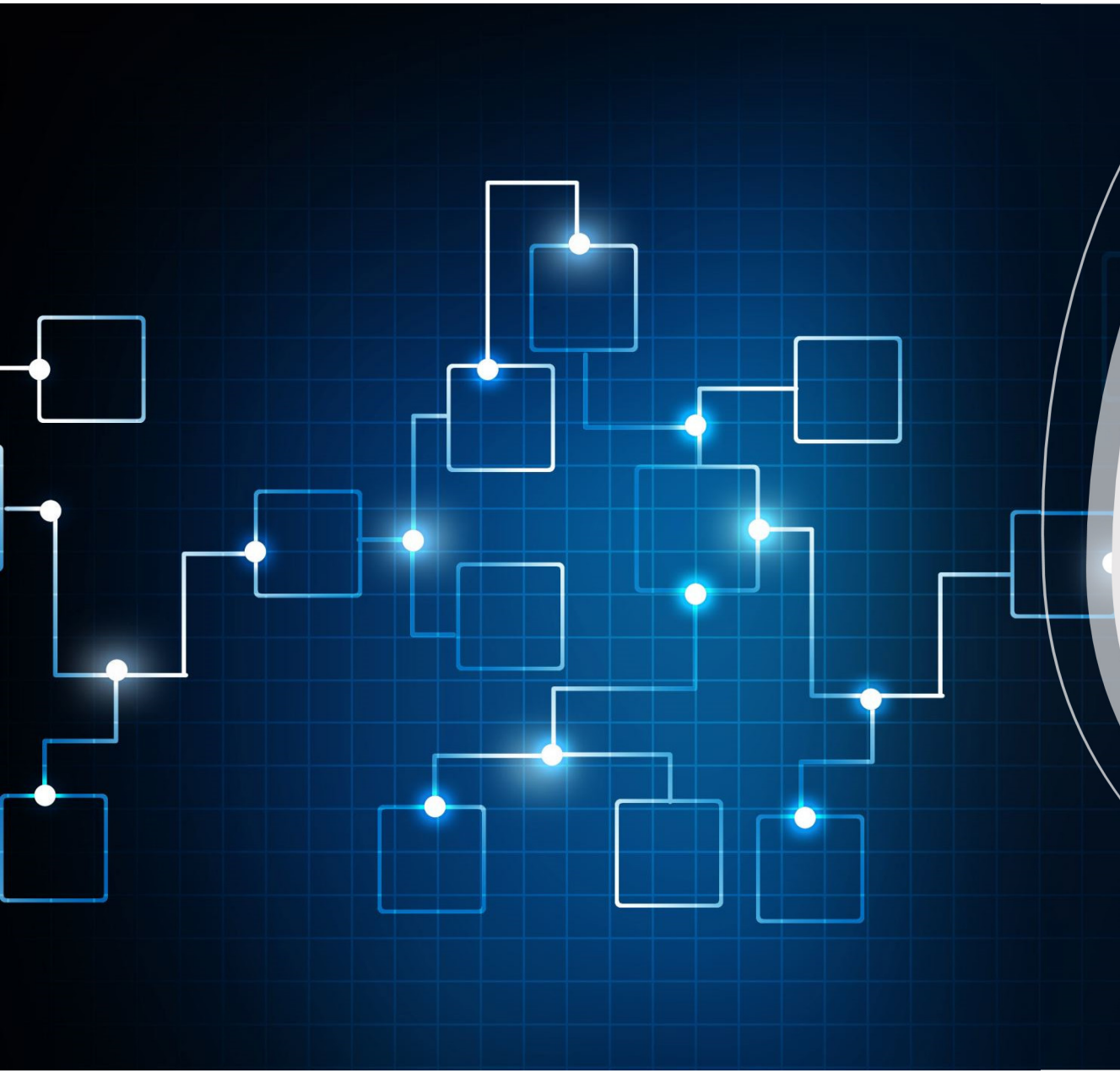
This choice is due to our work mechanism in the project was based on dividing the project into stages.

Each stage involved operations.

- Vision
- Planning
- Implementation
- Review
- Maintain
- Deployment

System analysis

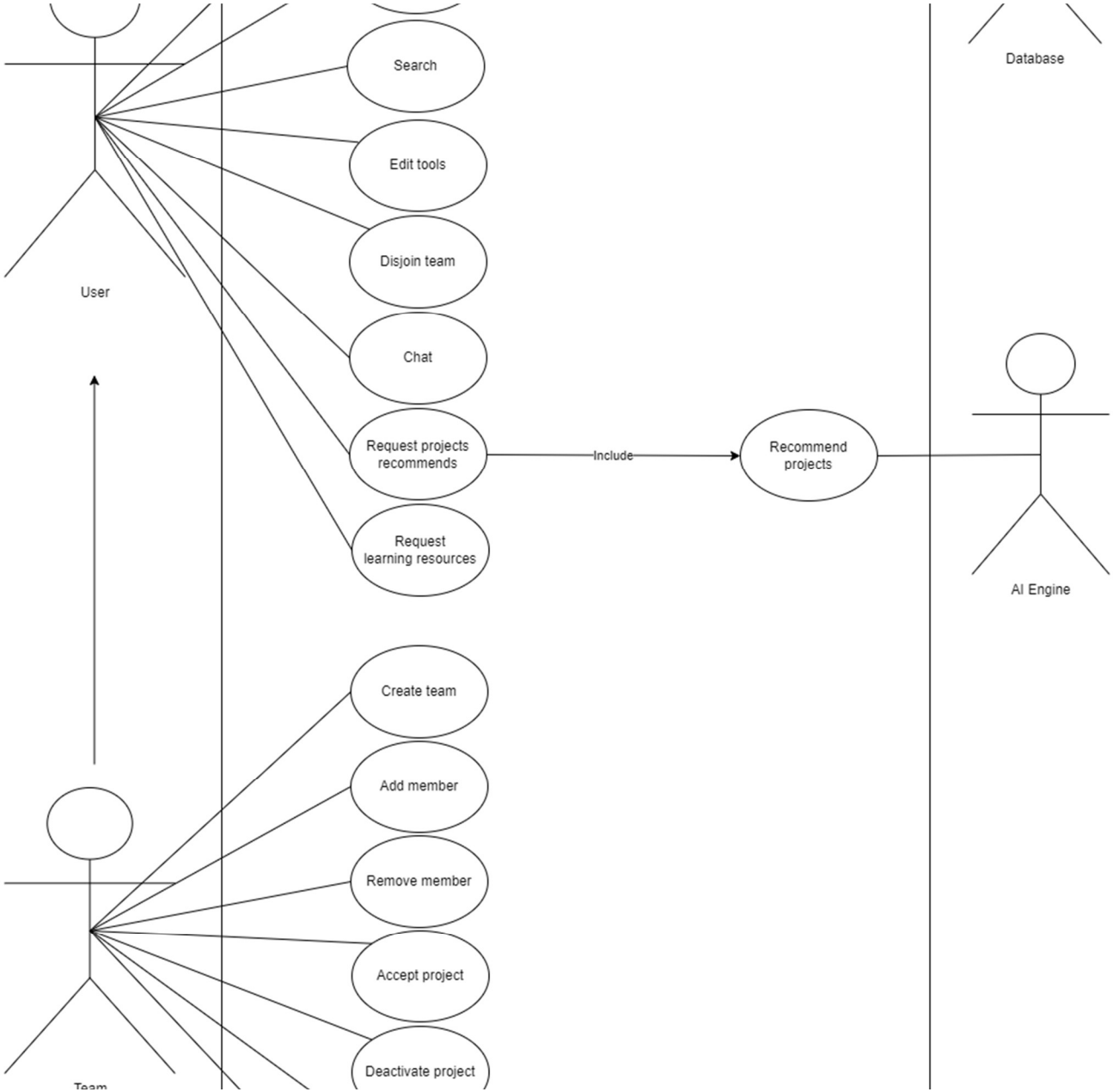




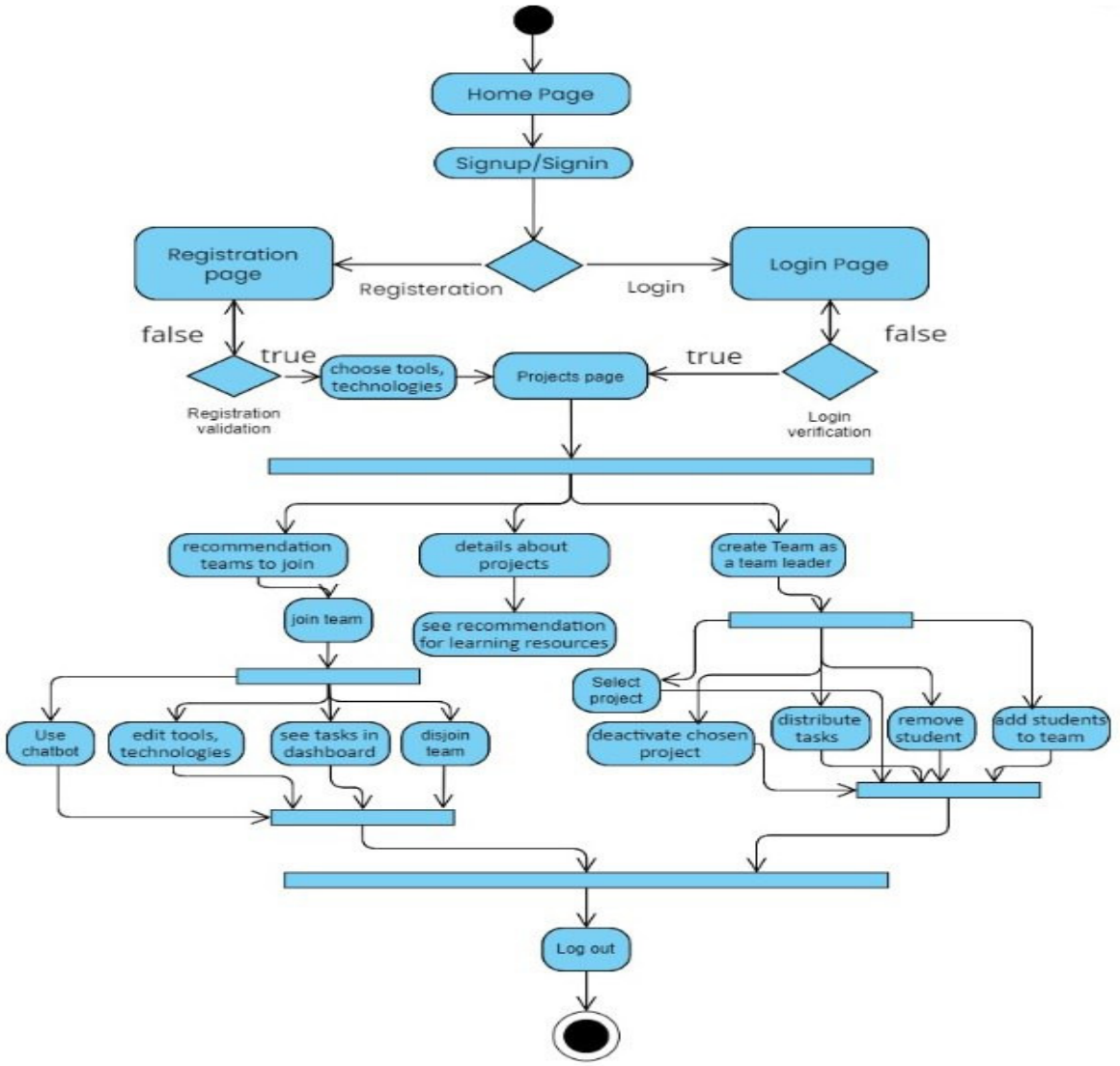
UML Diagrams

- **UML diagrams provide us by a standard language that communicates design information among software developers.**
- **UML diagrams allow us to see the big picture of a system.**
- **Our Diagrams include**
  - **Use case diagram**
  - **Activity Diagram**
  - **Sequence Diagram**
  - **Component Diagram**
  - **Class Diagram**
  - **Entity Relationship Diagram (ERD) & Mapping**

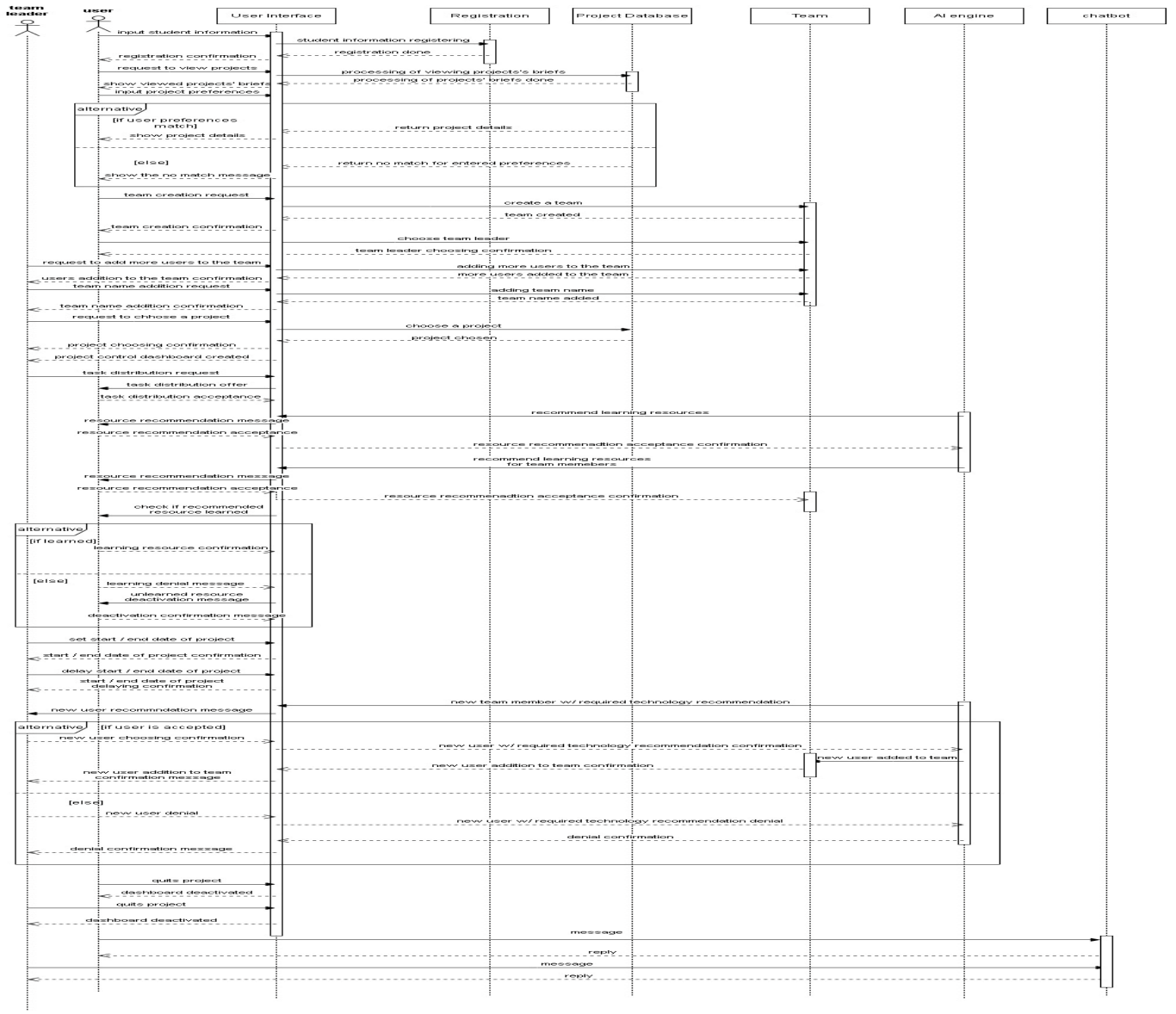
Use case diagram



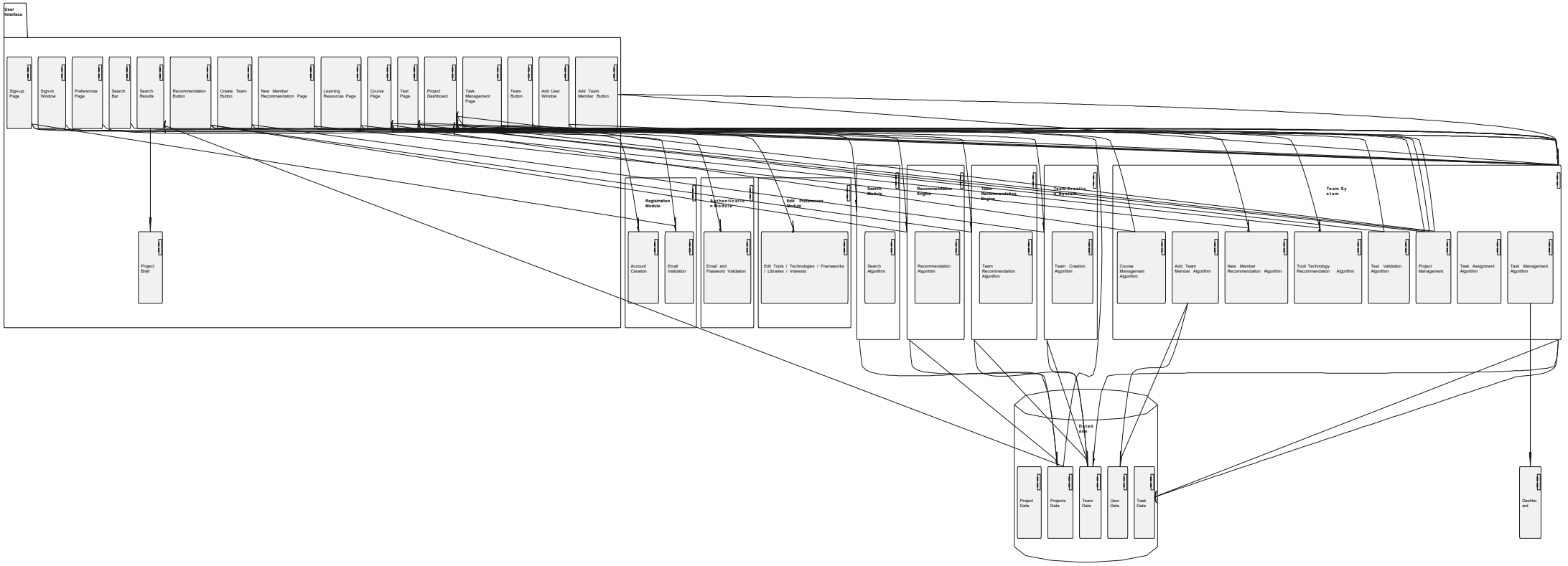
Activity diagram



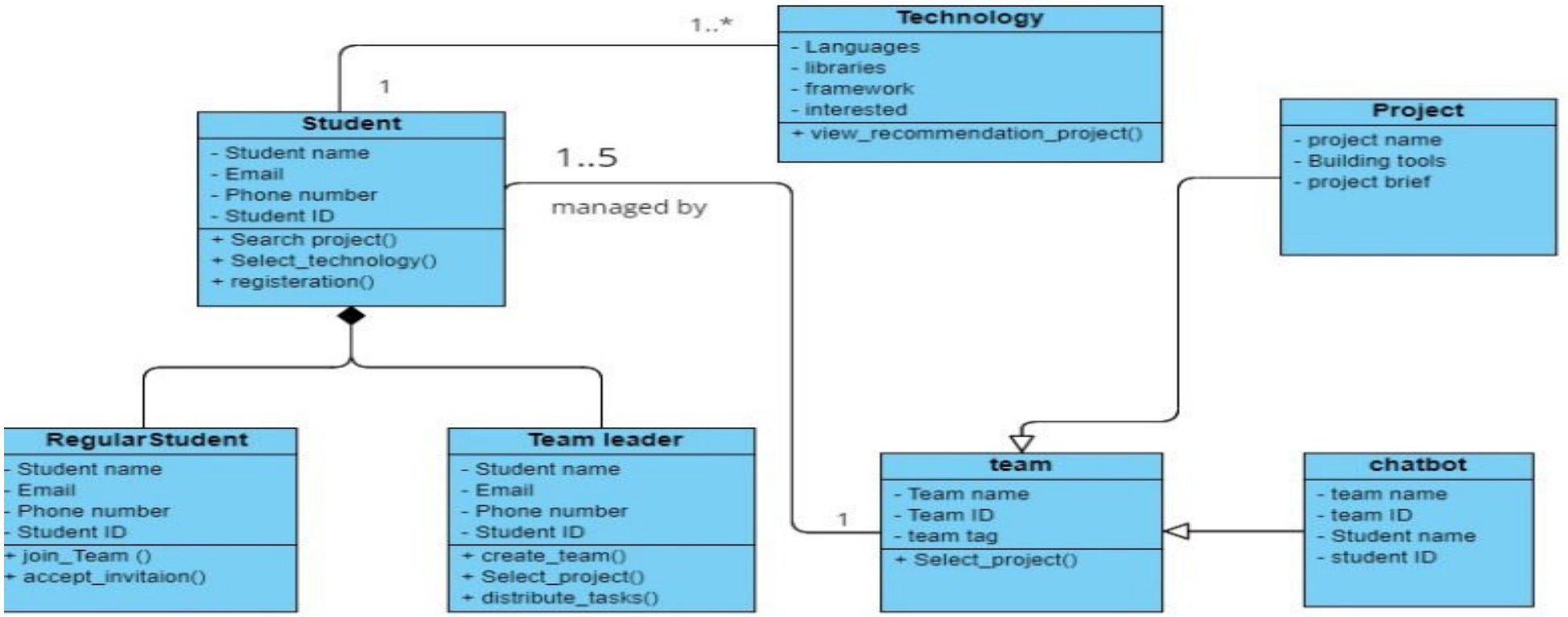
Sequence diagram



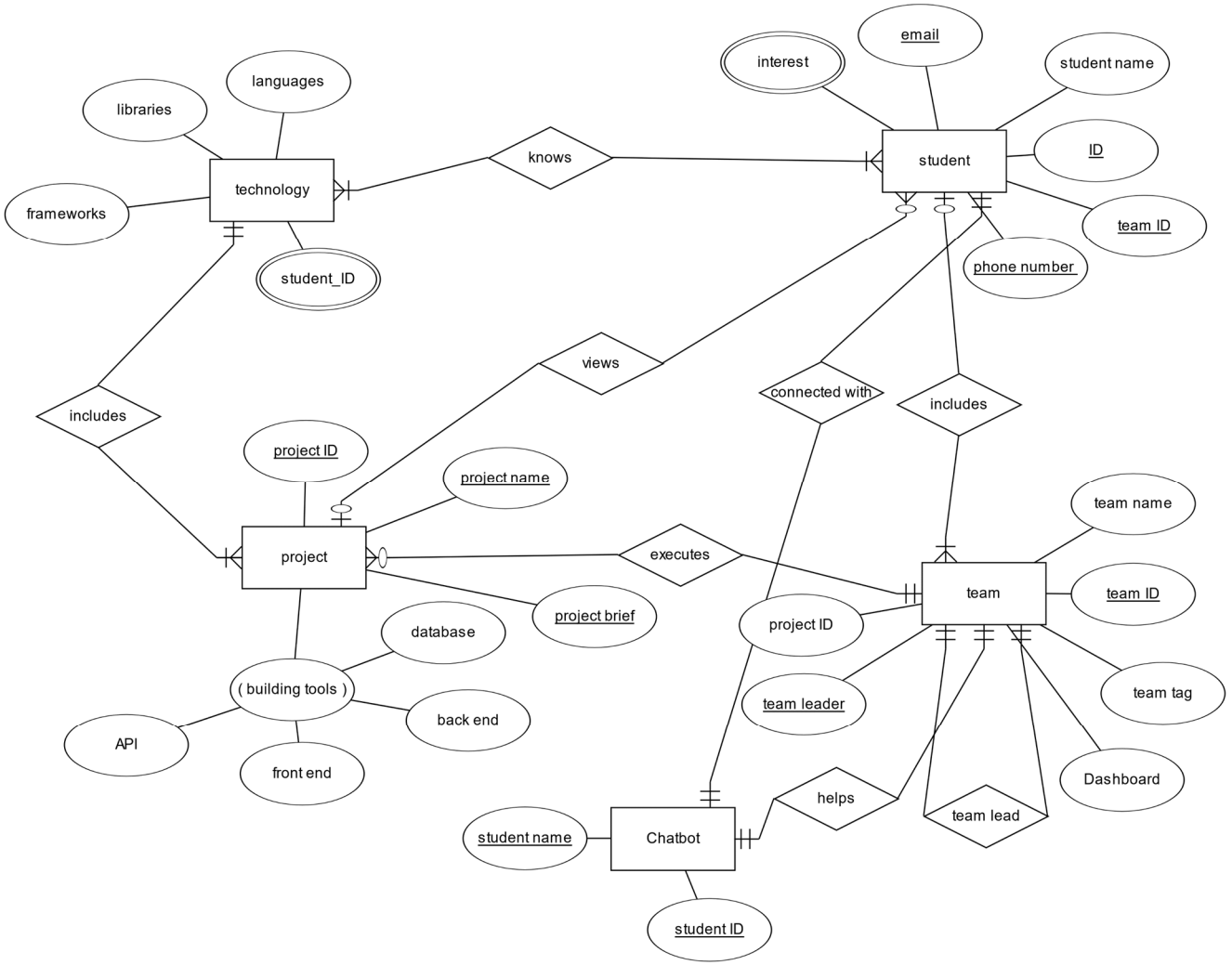
Component diagram



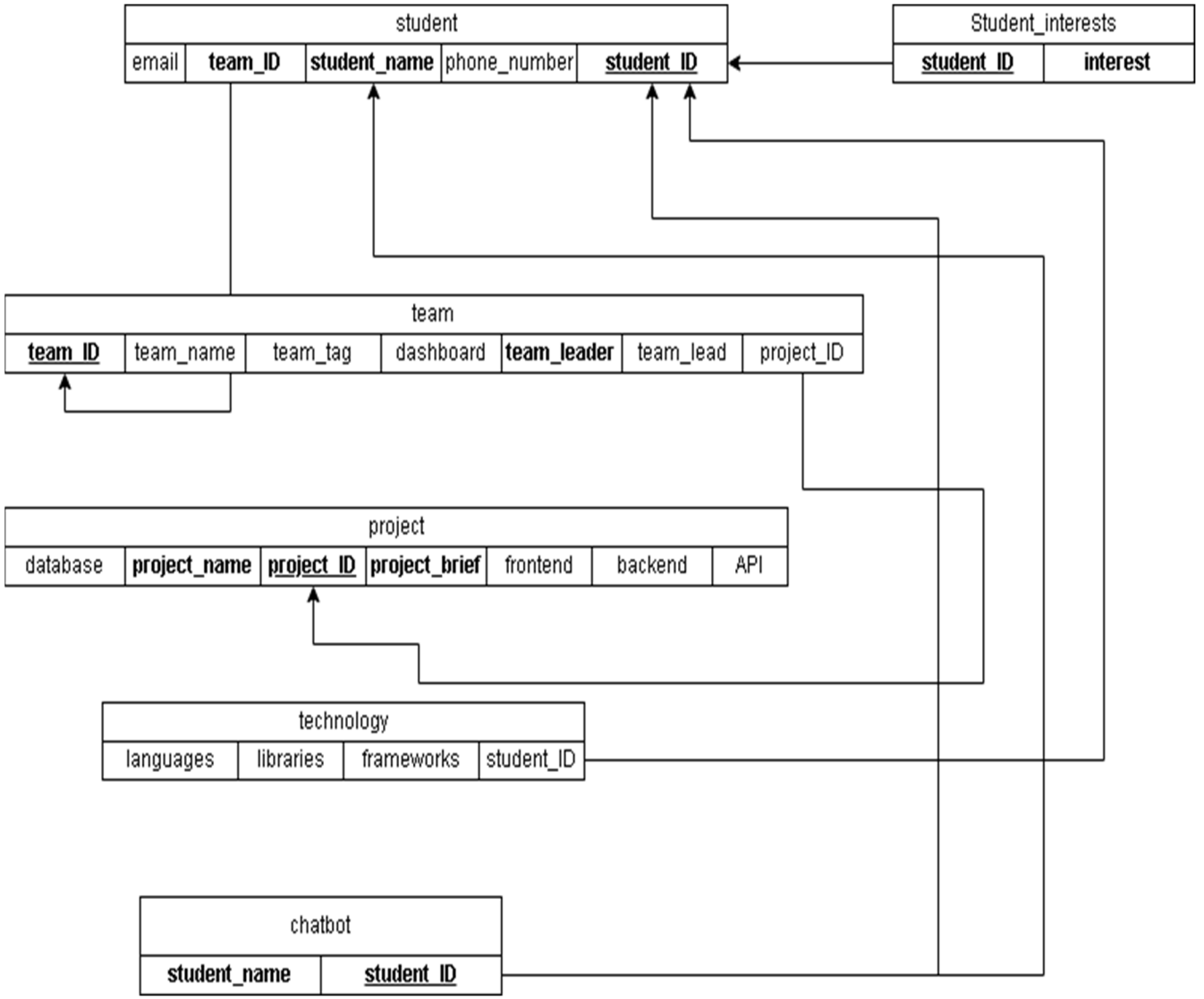
Class diagram



Entity Relationship Diagram (ERD)



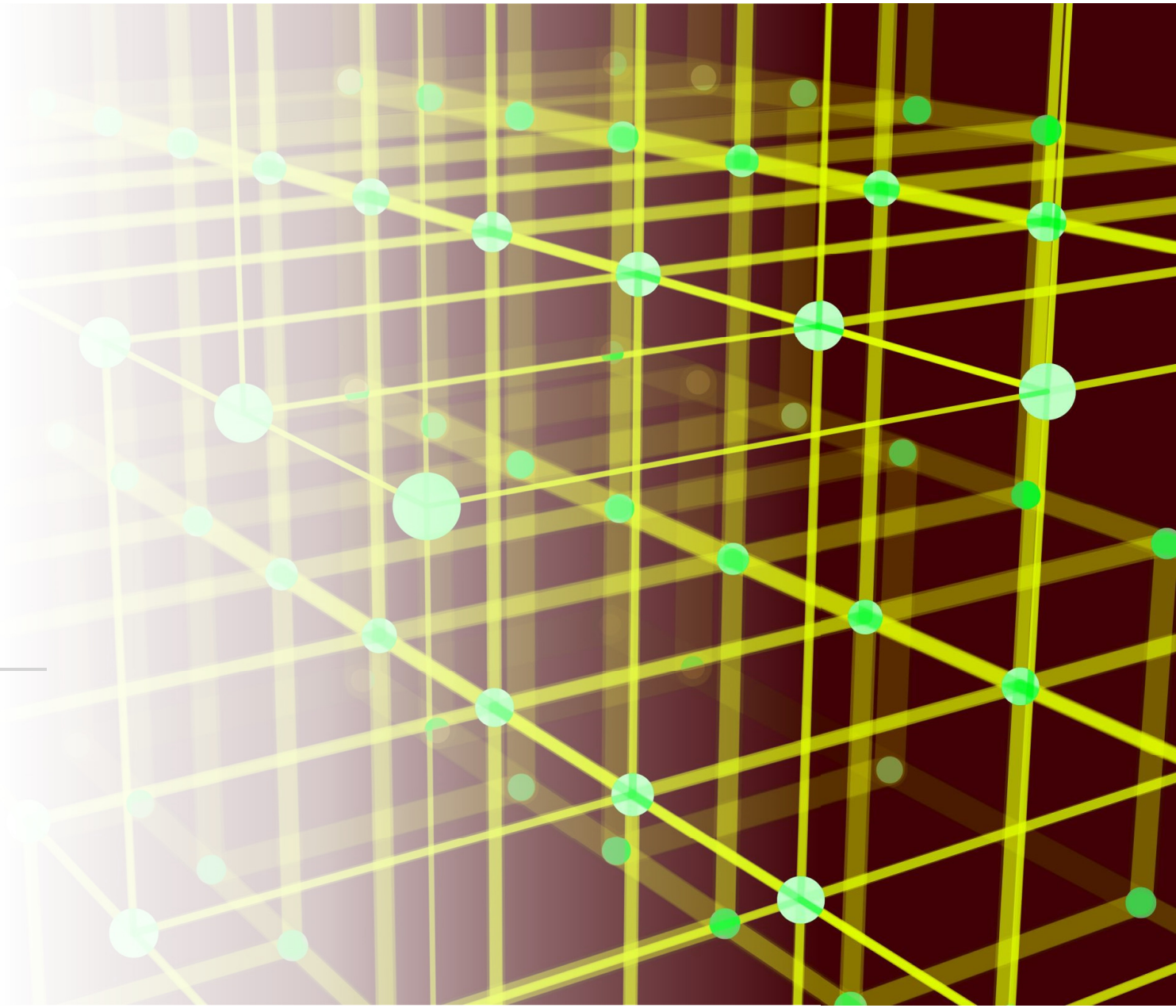
Mapping


student
email
team_ID
student_name
phone_number
student_ID
Student_interests
student_ID
interest
team
team_ID
team_name
team_tag
dashboard
team_leader
team_lead
project_ID
project
database
project_name
project_ID
project_brief
frontend
backend
API
technology
languages
libraries
frameworks
student_ID
chatbot
student_name
student_ID

Development Tools

- Front End
- Back end
- AI

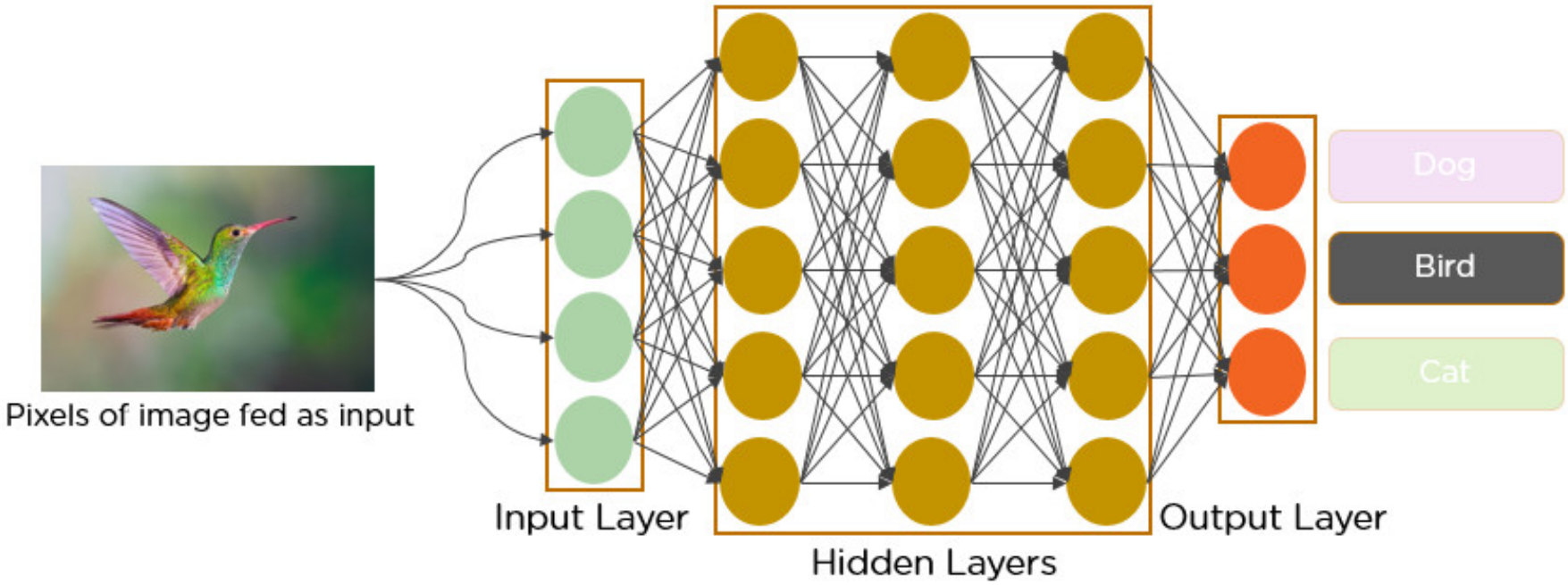
Interfaces



Backend

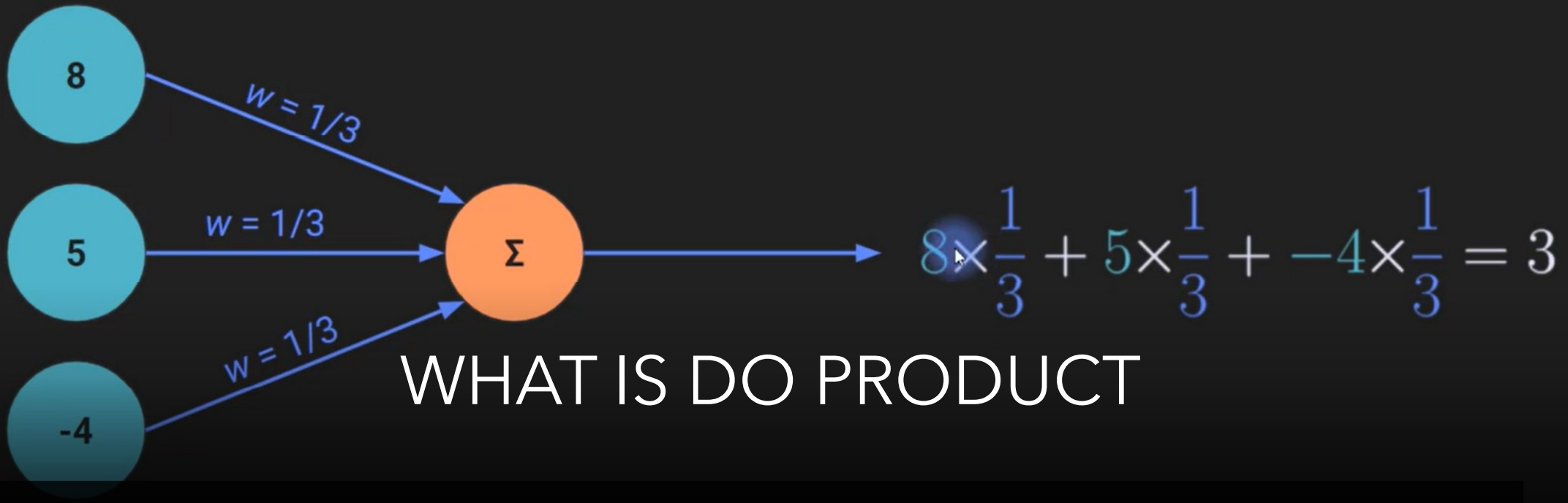

AUTOENCODER
MODEL

CLASSIFICATION PROBLEM



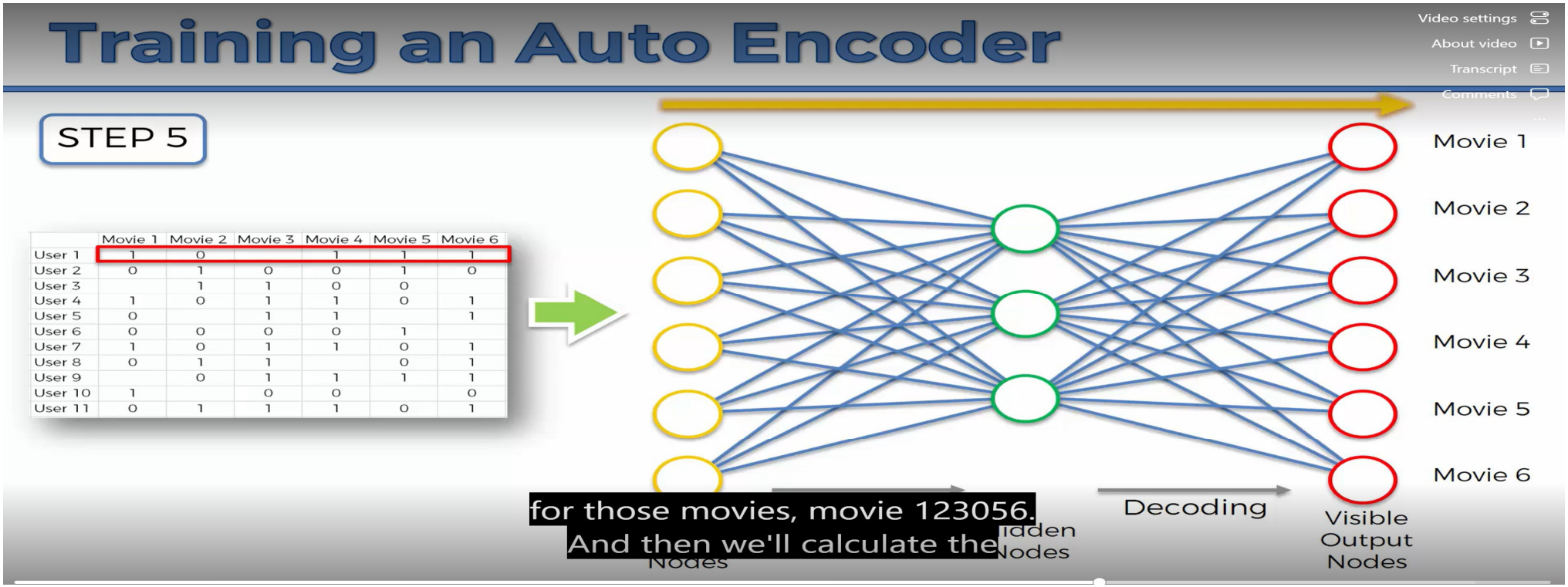
The perceptron

An averaging machine

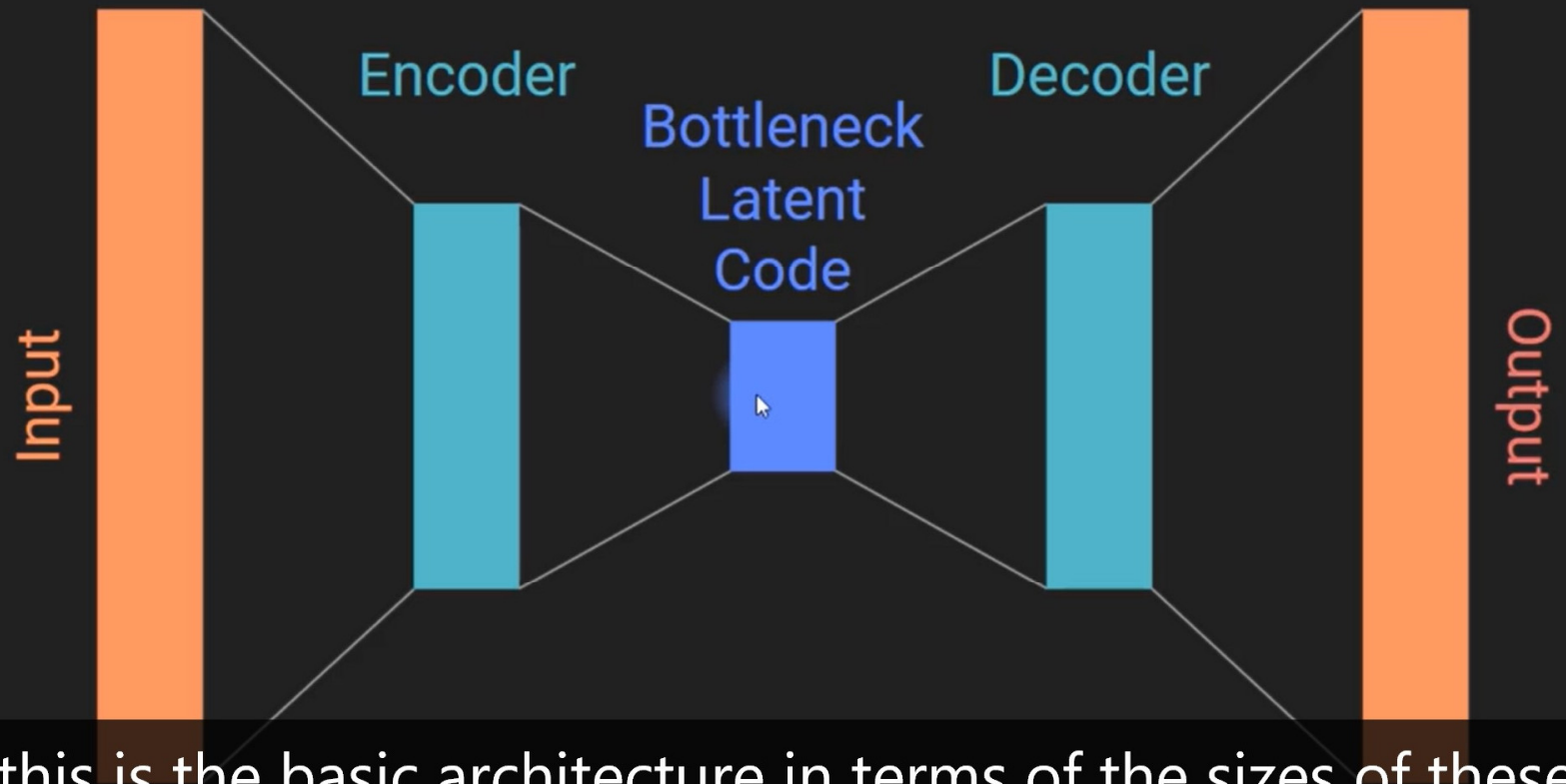


WHAT IS DO PRODUCT

By the way, notice what we're doing here with this mathematical operation.


Training an Auto Encoder
Video settings
About video
Transcript
Comments
STEP 5
Movie 1 Movie 2 Movie 3 Movie 4 Movie 5 Movie 6
User 1 1 0 1 1 1
User 2 0 1 0 0 1 0
User 3 1 1 0 0
User 4 1 0 1 1 0 1
User 5 0 1 1 1
User 6 0 0 0 0 1
User 7 1 0 1 1 0 1
User 8 0 1 1 0 1
User 9 0 1 1 1 1
User 10 1 0 0 0
User 11 0 1 1 1 0 1
Movie 1
Movie 2
Movie 3
Movie 4
Movie 5
Movie 6
for those movies, movie 123056.
And then we'll calculate the
Nodes
Decoding
Visible
Output
Nodes

Autoencoder architecture



OK, so this is the basic architecture in terms of the sizes of these layers.

WHAT IS AUTOENCODER

- The autoencoder has two main parts: the encoder and the decoder. The encoder takes in the input data and compresses it into a lower-dimensional representation, often referred to as a "latent code" or "embedding". The decoder then takes this compressed representation and reconstructs the original input data as closely as possible.

WHAT IS AUTOENCODER

An autoencoder is a type of neural network that learns to compress and then reconstruct data, typically images, but it can be applied to other types of data as well.

The goal of an autoencoder is to learn a compact representation of the input data that captures the most important features and patterns. This compressed representation can be used for various purposes such as data compression, dimensionality reduction, feature extraction, and even anomaly detection.

Why we use The SkillDataset class

01

The SkillDataset class is used to define a custom dataset for training the autoencoder model. The Dataset class provides a standard interface for accessing the data in a dataset, which makes it easy to use different datasets interchangeably in PyTorch.

02

In the case of an autoencoder, the input data and the target data are the same, since the goal is to reconstruct the input data as accurately as possible. Therefore, the __getitem__ method of the SkillDataset class returns a tuple containing the input data and the target data, both of which are the same.

03

By defining a custom dataset class, we can easily load and preprocess the data that we want to use for training the autoencoder. This can include tasks such as converting the data to the appropriate format (e.g., float), normalizing the data, and splitting the data into training, validation, and testing sets.

04

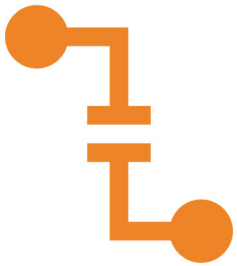
In summary, we use the SkillDataset class to define a custom dataset for training the autoencoder, which allows us to easily access and preprocess the data that we want to use for training.

AUTOENCODER ARCHITECTURE IN PYTORCH

The SkillDataset class is a custom dataset class that extends the Dataset class in PyTorch. It takes in a data parameter, which is a tensor containing the input data for the autoencoder. The __init__ method converts the input data to float format and stores it as an attribute of the class. The __getitem__ method returns a tuple containing the input data and the target data (which is the same as the input data, since this is an autoencoder). The __len__ method returns the length of the dataset.

The Autoencoder class is an autoencoder architecture that extends the nn.Module class in PyTorch. It takes in two parameters: input_size, which is the dimensionality of the input data, and hidden_size, which is the dimensionality of the compressed representation.

AUTOENCODER ARCHITECTURE IN PYTORCH



The __init__ method defines two linear layers: an encoder layer and a decoder layer. The encoder layer is a linear layer that takes in the input data and outputs the compressed representation. The decoder layer is another linear layer that takes in the compressed representation and outputs the reconstructed data.

The forward method specifies the forward pass of the autoencoder. It takes in an input tensor x, passes it through the encoder layer, applies the relu activation function, and passes the resulting compressed representation through the decoder layer. Finally, it applies the sigmoid activation function to obtain the reconstructed data.

KEY TERMS

num_epochs: The number of times to iterate over the entire dataset during training.

batch_size: The number of samples to process at once during each iteration (also known as a minibatch).

learning_rate: The step size to use when updating the model parameters during training.

input_size: The number of features in the input data (i.e., the number of columns in the training tensor).

hidden_size: The number of neurons in the hidden layer of the autoencoder.

KEY TERMS

The num_epochs parameter determines how many times the entire training dataset will be used to update the model weights. This is an important hyperparameter to tune, as too few epochs may result in an underfit model, while too many epochs may result in an overfit model.

The batch_size parameter determines how many samples are processed in each batch during training. Larger batch sizes typically lead to faster training times, but may also require more memory and can result in less stable updates to the model weights.

The learning_rate parameter determines the step size used to update the model weights during training. A higher learning rate may result in faster convergence, but can also cause the model to overshoot the optimal weights and result in a worse performance.

KEY TERMS

The input_size parameter is determined by the number of features in the input data. In this case, it is the number of columns in the training tensor, which corresponds to the number of programming skills in the dataset.

The hidden_size parameter determines the number of neurons in the hidden layer of the autoencoder. This is an important hyperparameter to tune, as it determines the complexity of the model and can affect both the training time and the quality of the reconstructed data.

SPLIT THE DATASET INTO TRAINING, VALIDATION, AND TEST SETS

- the data is first split into a training set (train_df) and a test set (test_df) using a test_size value of 0.2, which means that 20% of the data is used for testing and 80% is used for training.
- Then, the training set is further split into a new training set (train_df) and a validation set (val_df) using another test_size value of 0.2. This means that 20% of the training data is used for validation and the remaining 80% is used for actual training.

CREATE DATA LOADERS FOR TRAINING AND VALIDATION

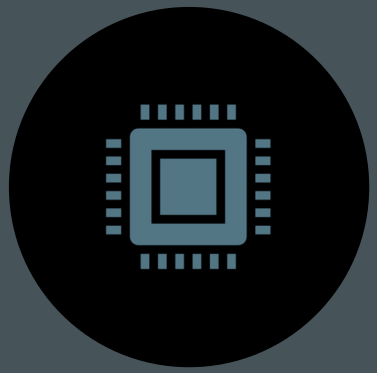
The DataLoader class is used to create an iterable dataset that can be used to iterate over the training and validation data in batches.

The shuffle parameter specifies whether to shuffle the order of the data samples during training. Shuffling the data can help prevent the model from overfitting to the order of the data, and can also help improve the convergence of the model.

For the training set, shuffle is set to True, which means that the data will be randomly shuffled during training. For the validation set, shuffle is set to False, which means that the data will not be shuffled during validation.

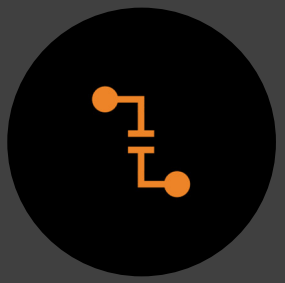
INITIALIZE THE AUTOENCODER AND OPTIMIZER

THE TORCH.OPTIM.ADAM FUNCTION IS USED TO CREATE AN OPTIMIZER THAT WILL BE USED TO UPDATE THE MODEL PARAMETERS DURING TRAINING. IT TAKES AS INPUT THE PARAMETERS OF THE AUTOENCODER MODEL (AUTOENCODER.PARAMETERS()) AND THE LEARNING RATE (LR), WHICH IS SET TO LEARNING_RATE.



THE TORCH.OPTIM.ADAM FUNCTION IS USED TO CREATE AN OPTIMIZER THAT WILL BE USED TO UPDATE THE MODEL PARAMETERS DURING TRAINING. IT TAKES AS INPUT THE PARAMETERS OF THE AUTOENCODER MODEL (AUTOENCODER.PARAMETERS()) AND THE LEARNING RATE (LR), WHICH IS SET TO LEARNING_RATE.

TRAIN THE MODEL



The outer loop iterates over the specified number of num_epochs, and the inner loop iterates over the batches of data in the train_loader.

For each batch of data, the inputs are extracted from the data loader, and the optimizer is reset to zero gradients. The autoencoder model is then used to generate outputs from the inputs, and the mean squared error loss between outputs and inputs is calculated using the criterion.

If the validation loss is better than the previous best loss, the model state dictionary is saved to the file autoencoder.pth.

At the end of training, the saved model state dictionary can be used to reconstruct the autoencoder model with the best weights learned during training.

EVALUATING THE PERFORMANCE OF THE TRAINED AUTOENCODER MODEL ON THE TEST SET

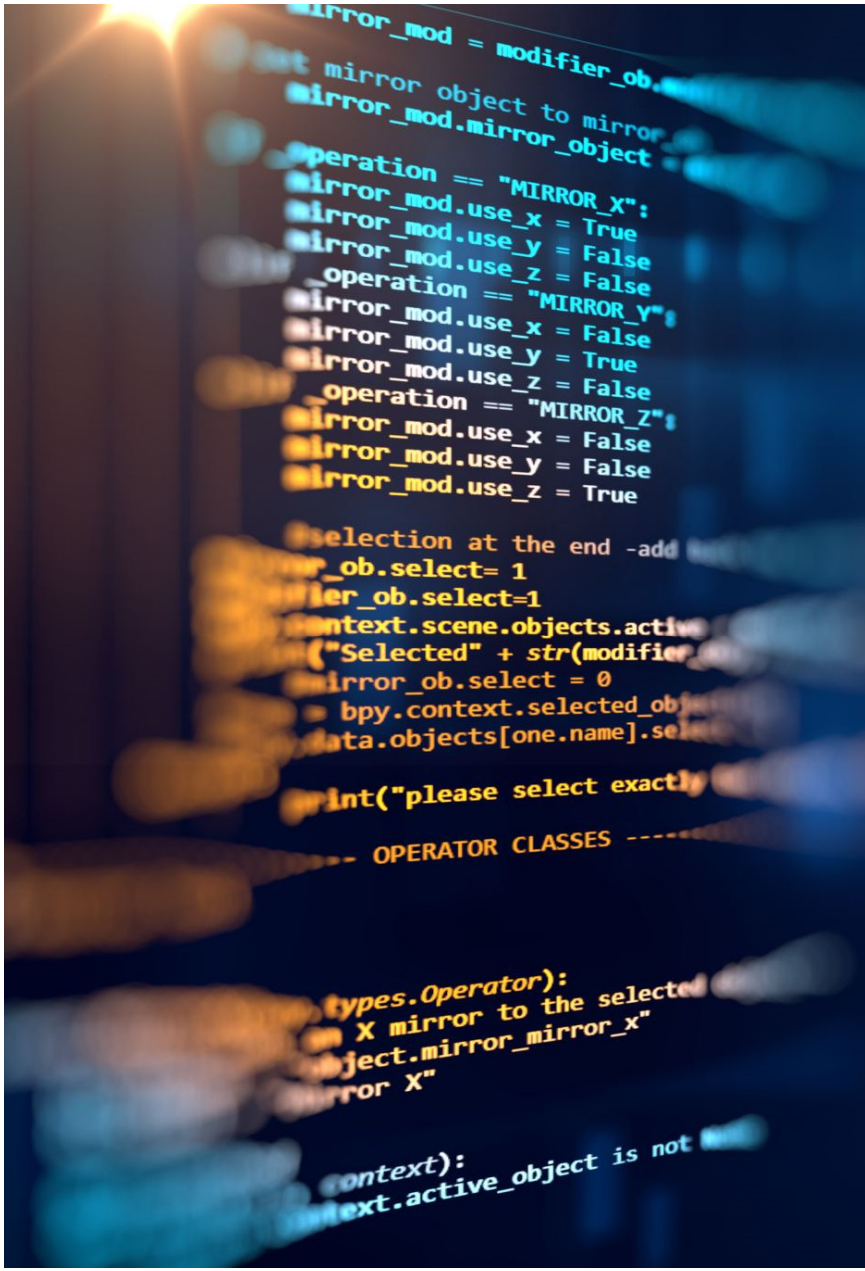
The saved weights of the best model are loaded using torch.load() and assigned to the autoencoder object using the load_state_dict() method.

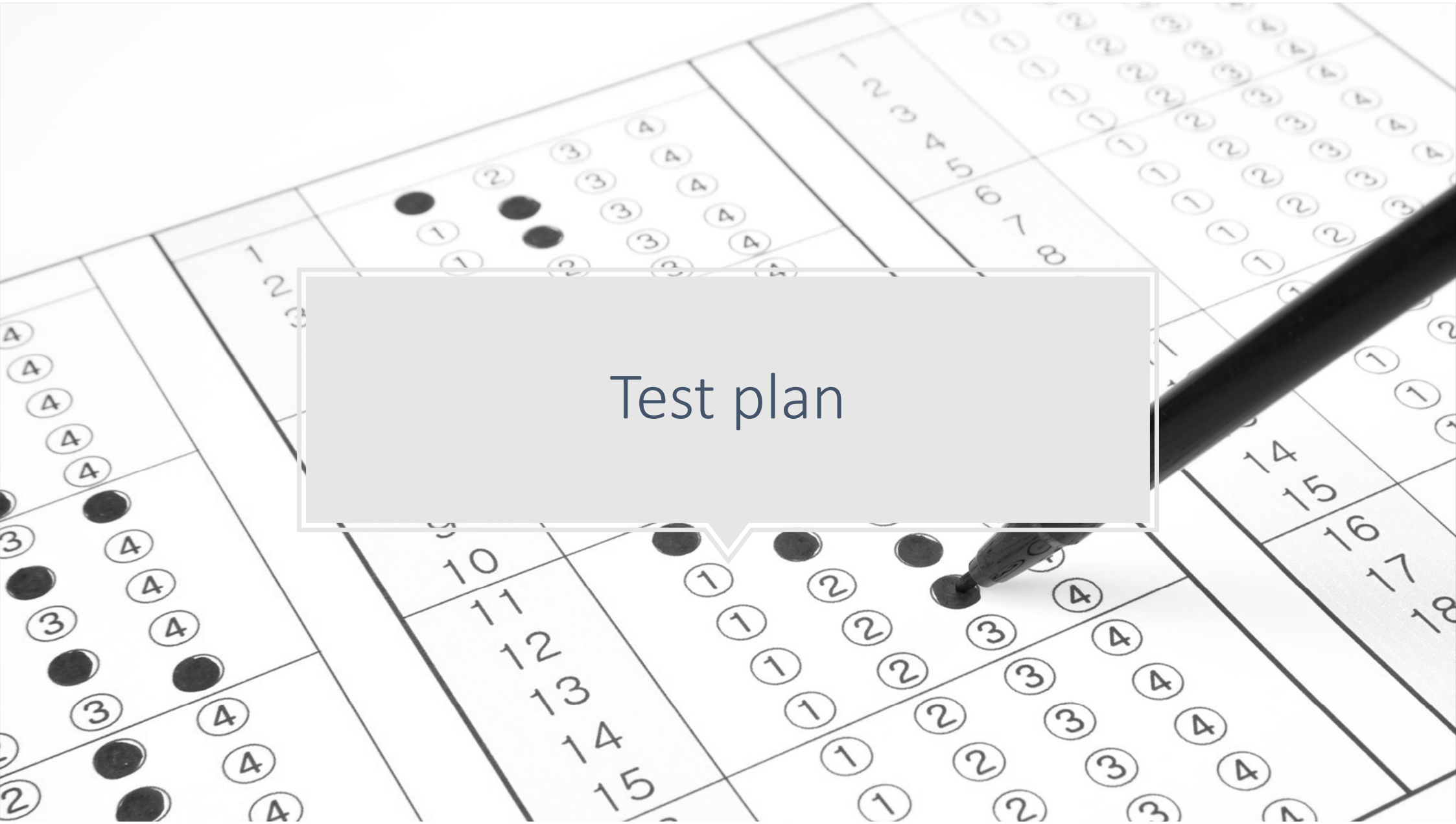
Then, a data loader is created for the test set using SkillDataset and DataLoader classes, similar to the training and validation sets. The test loss is calculated by iterating over the test loader, computing the reconstruction of each input using the autoencoder, and comparing it with the original input using the mean squared error loss. The average loss over the entire test set is then printed.

By evaluating the model on a previously unseen test set, we can determine the generalization performance of the autoencoder and ensure that it has not overfit to the training set.

ENCODE THE PROGRAMMERS AND PROJECTS INTO A LOWER-DIMENSIONAL SPACE

- The code encodes the programmers' and projects' skill data into a lower-dimensional space using the trained autoencoder. Then, it computes the cosine similarity between each programmer and each project in the encoded space, and finds the most similar project for each programmer. Finally, it prints the recommendations for each programmer by printing the recommended project.




Test plan

Conclusion and Future work