TED UNIVERSITY

CMPE 492

Senior Design Project II

VENATOR

Low-Level Design Report

10/03/2024

Team Members:

- **Arda Uyaroğlu**
- **Berker Tomaç**
- **Selçuk Akif Topkaya**
- **Utku Oktay**

Table of Contents

1 Introduction

This report aims to propose the low-level design of Venator, a detect & track service that provides statistics for various components and events in football matches. Venator applies computer vision techniques and various machine learning algorithms to the video recording of football matches in order to detect and track the key components of a match (namely the players, goalkeepers, referees and the ball) and it produces some statistics about players, teams, and the match as a whole (such a tracking player positions, calculating distances traveled, determining average player positions, generating heat maps, differentiating the players of the opposing teams, and computing the ball possession duration of each team, among other features.) as a result of its analyzation, thus, helps the football couches and stakeholders of the teams with their decisions to improve their players and their teams.

This repost provides a comprehensive look at the low-level design of Venator, stating its object design trade-offs, the engineering standards that are practiced during the development, its interface documentation guideline, the interfaces and packages that are (and will be) used during the development.

In essence, this document delves into the low-level design elements and methodologies employed by Venator to achieve its objectives, providing a detailed blueprint for the implementation and deployment of this innovative service.

1.1. Object Design Trade-offs

1.1.1. Usability vs. Complexity

Venator is built with providing easily usable interfaces in mind. However, maintaining ease of use and simplicity may reduce the provision of enhanced functionalities within the app with advanced customization options and complex features.

1.1.2. Reliability vs. Scalability

Venator aims to provide highly accurate information in its statistics that it provides after the analyzation of football matches. Achieving high level reliability in the

provided statistics may increase the computational costs and optimization costs, thus, reduce the system's overall scalability.

1.1.3. Security vs. Performance & User Experience

Venator employs a secure authentication and verification service that protects the user data from activities of malicious parties. However, these measures may introduce additional overhead, thus, may affect performance and user experience.

1.1.4. Availability vs. Operational Costs

Venator aims for high availability by providing an application that is easily accessible via web from any location, without the need of any installments to desktops or other devices. However, promising ease of availability may introduce the need to use proper host services and maintain a dynamic & permanent server, thus, may increase operational costs.

1.2. Interface Documentation Guidelines

In this report, the following documentation style is used for the classes in the Class Interfaces section:

Class:	Class Name
Description:	Class Description
Attributes:	Class Attributes
Methods:	Class Methods

Table 1: Interface Documentation Guideline

1.3. Engineering Standards

Venator's design and documentation processes are carried out in a manner compatible with established engineering standards in order to preserve consistency and professionalism throughout the report.

The report follows the Unified Modeling Language (UML) guidelines for the visualization purposes of designing diagrams, interfaces, subsystem decompositions, and the IEEE Standards for use of language and formatting throughout the report.

By complying with these standards, the report preserves clear and structed visualizations and a professionally written report as a whole.

1.4. Definitions, acronyms and abbreviations

UML: Unified Modeling Language

IEEE: Institute of Electrical and Electronics Engineers

GUI: Graphical User Interface

CRUD: Create, read, update and delete (four basic operations of persistent storage in database management systems)

2 Packages

This section provides the packages for Venator's GUI and Back-End, respectively. Venator has two distinct packages for its front-end (GUI) and back-end. Front-End package is responsible for the displaying of the web application, while the Back-End package is responsible for the authentication and verification of the users, database's CRUD operations, creation, viewing and analyzation of the game records and computing various statistics for the key components of a game. The services that these distinct packages implement work with unison to create the desired final application and cooperatively manage its operations.

2.1. GUI Packages

GUI Package provides the interactions that can take place between the end-users and the web application. Users can signup or (if they already have an account) login to the application via their respective interfaces. They can reset their passwords if they forget it. They can create new game records for the application to analyze and provide statistics for. Users can view the past game records. They can also view their account information page and can edit their information as well.

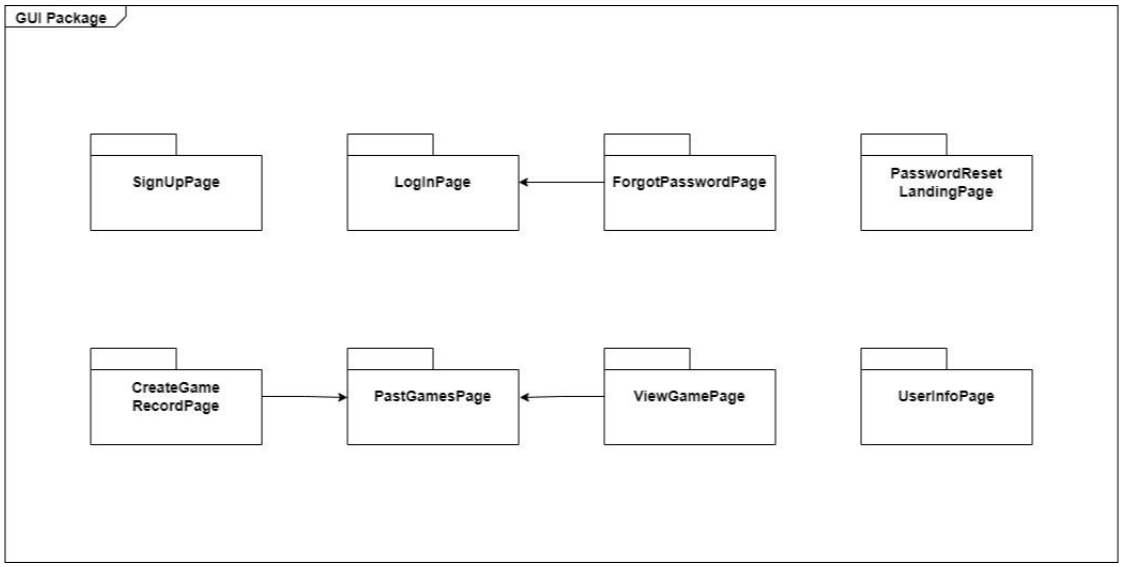

Figure 1: GUI Package

2.2. Back-End Packages

Back-End package is responsible for communication with the front-end side, handling the database and calculation of analytics regarding the uploaded games. Computing the analytics/statistics involves several pre-analyses such as object and keypoint detection and tracking, homography estimation for finding 2D projection of the field and color masking for team identification. We use machine learning algorithms for these tasks as ML models are more successful at carrying out such tasks than conventional techniques.

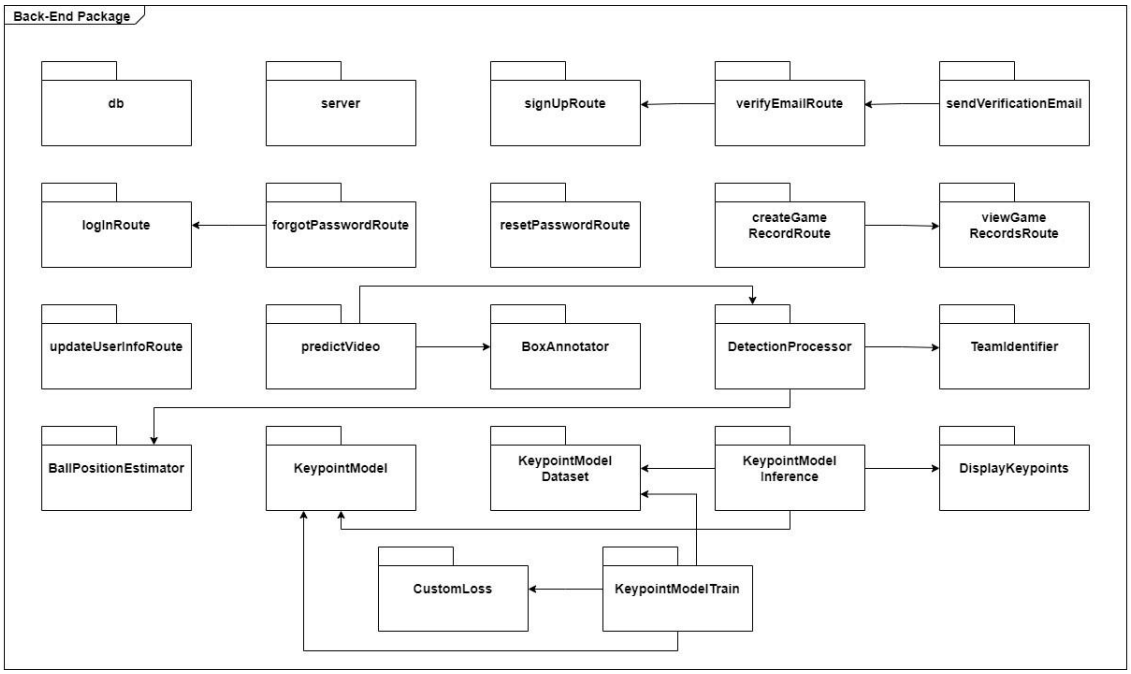

Figure 2: Back-End Package

3 Class Interfaces

3.1. GUI Classes

Class:	SignUpPage
Description:	This class manages the displaying of the signup page on web
Attributes:	Email password
Methods:	onSignUpClicked()

Class:	LogInPage
Description:	This class manages the displaying of the login page on web
Attributes:	email password
Methods:	onLogInClicked()

Class:	ForgotPasswordPage
Description:	This class manages the displaying of the forgot password page on web
Attributes:	email
Methods:	onSubmitClicked()

Class:	PasswordResetLandingPage
Description:	This class manages the displaying of the password reset landing page on web
Attributes:	newPassword
Methods:	onResetClicked()

Class:	CreateGameRecordPage
Description:	This class manages the displaying of creating a game record on web
Attributes:	gameRecordName gameVideoRecording teamNames teamJerseyColors
Methods:	onCreateRecordClicked()

Class:	PastGamesPage
Description:	This class manages the displaying of the past games on web
Attributes:	games
Methods:	displayPastGames()

Class:	ViewGamePage
Description:	This class manages the displaying of viewing a game record on web
Attributes:	gameRecordId gameRecordInfo gameStatistics
Methods:	displayGameRecord()

Class:	UserInfoPage
Description:	This class manages the displaying of the user info page on web
Attributes:	user userId userInfo
Methods:	editUserInfo() saveChanges() logOut()

3.2. Back-End Classes

Class:	db
Description:	This class manages the creation of the database connection
Attributes:	db client
Methods:	connectToDb()

Class:	server
Description:	This class manages the creation of the endpoints for all routes
Attributes:	app port
Methods:	initializeConnection()

Class:	sendVerificationEmail
Description:	This class manages the creation and serving of the verification emails
Attributes:	subject content sender to
Methods:	sendEmail()

Class:	signUpRoute
Description:	This class manages the signup logic of the web application (creation of a new user and encryption of their password)
Methods:	handler()

Class:	logInRoute
Description:	This class manages the login logic of the web application by using secure measures with json web tokens and comparisons of the hashed passwords
Methods:	handler()

Class:	verifyEmailRoute
Description:	This class manages the verification of the emails when users signup to the web application
Methods:	handler()

Class:	forgotPasswordRoute
Description:	This class manages the sending of password reset emails when users forget their passwords
Methods:	handler()

Class:	resetPasswordRoute
Description:	This class manages the resetting of passwords with a new value of the user's choosing
Methods:	handler()

Class:	createGameRecordRoute
Description:	This class manages the creation of a new game record on the database
Methods:	handler()

Class:	viewGameRecordsRoute
Description:	This class manages the viewing of a game record via the bridge between front-end and back-end
Methods:	handler()

Class:	updateUserInfoRoute
Description:	This class manages the updating of user informations
Methods:	handler()

Class:	predictVideo
Description:	This class manages object detection on the video when a YOLO model and path to video file is provided
Attributes:	model classColors imgsz boxAnnotator tracker detectionProcessor
Methods:	perform_detection() annotate_frame() analyze_video() predict_video()

Class:	BoxAnnotator
Description:	This class manages annotation of the provided frame by receiving detections and drawing rectangles accordingly
Attributes:	box_colors
Methods:	annotate()

Class:	DetectionProcessor
Description:	This class manages processing of the detections provided by YOLO model with the help of BallPositionEstimator and TeamIdentifier classes

Attributes:	_detections teamIdentifier
Methods:	read_detections_from_csv() write_detections_to_csv() _visualize_ball_movement() _symmetric_moving_average() butter_lowpass_filter() _apply_low_pass_filter_to_ball() _find_ball_center() _euclidian_distance_between_ball_positions() _distance_between_ball_and_line() _find_frame_with_one_ball() _find_closest_candidate() _interpolate_position() interpolate_ball() getDetections() addDetection()

Class:	**TeamIdentifier**
Description:	This class manages the identification of the team of the player when a cropped image of the detected player is provided
Attributes:	classColors
Methods:	identifyTeam()

Class:	**BallPositionEstimator**
Description:	This class manages estimation of the position of the ball by making use of graph optimizations and various signal processing techniques
Attributes:	G ball_positions detected_to_undetected_weight undetected_to_detected_weight undetected_to_undetected_weight
Methods:	extract_ball_positions_from_detections() _distance_between_two_positions() _weight_between_two_nodes() build_ball_graph() find_shortest_path()

Class:	KeypointModel
Description:	This class is a PyTorch artificial neural network model inheriting from torch.nn.Module that approaches key detection problem as a regression problem
Attributes:	fc feature_extractor
Methods:	forward()

Class:	KeypointModelDataset
Description:	This class manages fetching desired data from the dataset on the disk and inherits from torch.utils.data.Dataset
Attributes:	imgs transform root annotations
Methods:	_read_annotations() _get_annotations_of_image() __len__() __getitem__()

Class:	KeypointModelInference
Description:	This class is used for testing the performance of KeypointModel by performing inference on the validation dataset and visualizing the predictions on the image by using DisplayKeypoints class to allow visual inspection of the model's performance
Attributes:	display_keypoints model data_loader device dataset
Methods:	infer()

Class:	DisplayKeypoints
Description:	This class manages visualization of the provided keypoints on the given image
Methods:	display()

Class:	CustomLoss
Description:	This class, which inherits from torch.nn.Module, is used to penalize the KeypointModel during training and to measure its performance by combining Mean Squared Error and Binary Cross Entropy loss functions as model's output involve both regression and classification values and the model must not be penalized for misprediction of the keypoints that are not visible in the image
Attributes:	mse_loss factor bce_loss
Methods:	forward()

Class:	KeypointModelTrain
Description:	This class manages the training of the KeypointModel
Attributes:	device
Methods:	train() train_model() evaluate_model() print_gpu_memory_usage()

4 Glosarry

Host Service: A web hosting service. A type of Internet hosting service that hosts websites for clients.

Engineering Standards: Formal technical documents that establish uniform engineering norms, methods, practices or requirements.

Front-End: The part of a website that the end-users visually and physically interact with.

Back-End: The part of a website (which cannot be accessed by the end-user) that manages the operations on mostly data-driven logic and server-client interactions, e.g. databases, URL endpoints, user authentication and verification for security purposes, etc.

Color Masking[1]: A technique where an image is transformed into a binary format based on a specified range of colors in order to extract certain objects or parts of the image based on their color.

Homography Estimation[2]: A technique used in computer vision and image processing to find the relationship between two images of the same scene but captured from different viewpoints. It is used to align images, correct for perspective distortions or perform image stitching.

5 References

1. Riswanto, U. (2023, February 20). A beginner's guide to image segmentation using color masking. Medium. https://ujangriswanto08.medium.com/a-beginners-guide-to-image-segmentation-using-color-masking-3a3fd536ed25#:~:text=Color%20masking%20is%20a%20technique,image%20based%20on%20their%20color.
2. Homography estimation. Papers With Code. (n.d.). https://paperswithcode.com/task/homography-estimation#:~:text=Homography%20estimation%20is%20a%20technique,distortions%2C%20or%20perform%20image%20stitching.
3. NASA Office of Logic Design. 4.1 Design Tradeoffs. (n.d.). https://klabs.org/history/history_docs/sp-8070/ch4/4p1_design_tradeoffs.htm