**University of Karachi**

**Department of Computer Science**

****

**Final Year Project Report**

Vehicle/Automobile Parking Security System

**Supervise by:**

**Sir Dr. Tehseen Jilani / Sir Usman Amjad**

**Submitted by:**

**Parkash Kumar (B11101067)**

**S .Haider Ali Naqvi (B10101109)**

**Waqar Younus (B10101103)**

ABSTRACT:

Automotive theft has been a persisting problem all around the Pakistan especially in our Karachi and greater challenge comes from professional thieves. In this paper, we present an automobile parking security system.

This security system is based upon machine vision / image processing technology that takes it to a new level of theft protection. It is based on hardware system, for real time acquisition of number-plate images using an active IR illuminator. This system can locate and recognize the number plate, identify the unauthorized number plate. When the unauthorized car arrives, our system will alarm and gives the option for registration of the vehicle. For being more secure we had made this live streaming as well, so that the police can see/search any vehicle in our parking area through CDMA or GPRS networks. The status of the parking field detected by sensor nodes is reported periodically to a database via the deployed wireless sensor network and its gateway. The database can be accessed by the upper layer management system to perform various management functions, such as finding vacant parking lots, auto-toll, security management, and statistic report.The system was tested in a simulating environment and it was found very robust, reliable. To prove the effectiveness of the system proposed by us we have developed and presented a mathematical model which will be discussed in brief further in the paper.

**Acknowledgement:**

I would like to express my gratitude and appreciation to all those who gave me the possibility to complete this project. A special thanks to our final year project coordinator, Sir Usman Amjad, whose help, stimulating suggestions and encouragement, helped me to coordinate my project especially in writing this report.

I would also like to acknowledge with much appreciation the crucial role of the staff of computer lab and library, who gave the permission to use all required books and the necessary material to complete this project

A special thanks goes to my team mate, Syed Ali Haider Naqvi and Waqar Younus, who helps me to complete the project and gave suggestion and innovative ideas about this project

Last but not least, many thanks go to the head of the project, Dr. Tehseen Jilani whose have given his full effort in guiding the team in achieving the goal as well as his encouragement to maintain our progress in track. I would to appreciate the guidance given by other supervisor as well as the panels especially in our project presentation that has improved our presentation skills by their comment and tips.

Table of Content

**Content Page**

1. Introduction …………………………………………………................... 6

1.1 Project Motivation ………………………………………………… 6

1.2 Purpose of the System …………………………………………….. 6

1.3 Scope of the System ………………………………………………. 6

1.4 Project Objectives …………………………………………………. 7

1.5 Current System …………………………………………………….. 7

1.6 Proposed System …………………………………………………… 7

1.7 Significance …………………………………………………………. 7

1.8 Background ………………………………………………………….. 8

2**.** SDLC………………………………………………………………………….. 11

2.1 Waterfall Model………………………………………………………. 12

3**.** Image Processing…………………………………………………………… 14

3.1 Background………………………………………………………… 14

3.2 What is Image Processing?..................................................... 15

3.3 Edge Detection……………………………………………………. 16

3.4 Text Localization Method…………………………………………17

3.5 Other Methods……………………………………………………. 20

4**.** Automate Process……………………………………………………….. 24

4.1 Gulp - The task runner………………………………………….. 24

4.2 How to use Gulp?............................................................... 25

4.3 Why we use task runner?..................................................... 25

4.4 Why use Gulp?................................................................... 25

5. Project Overview……………………………………………………….. 26

5.1 Registration panel………………………………………………. 27

5.2 Logs……………………………………………………………… 29

5.3 Entries…………………………………………………………… 33

5.4 Live streaming………………………………………………….. 36

5.5 Notifications……………………………………………………. 38

5.6 Task runner……………………………………………………… 39

5.7 Database………………………………………………………… 41

5.8 Image processing using Matlab……………………………….. 44

6. Description of Hardware System…………………………………….. 46

6.1 Camera…………………………………………………………. 46

6.2 Computer/Laptop……………………………………………… 46

7**.** Description of Technologies………………………………………….. 46

7.1 Matlab………………………………………………………….. 46

7.2 Nodejs………………………………………………………….. 48

7.3 Ajax…………………………………………………………….. 59

7.4 Mongodb……………………………………………………….. 66

8**.** Conclusion……………………………………………………………… 80

7**.** References………………………………………………………………. 81

# **Introduction**

# **1.1 Project Motivation.**

In many countries parking is one of the major issues and this problem is mostly faced by most of the Asian countries especially Pakistan is one which severely surround by this problem.

In Pakistan and specific in Karachi the rate of this problem is increasing which affect the rate of terrorism activities and now this problem is also facing by many universities our university is one of it.

We want to develop a “Car Parking Security System”, which trace the number plate of a car and perform the following task

1. Verify the number of a car according to records University whether it is registered or not
2. Notify the head of all monitors through email.

**1.2 Purpose of the System.**

The main theme of our project is to develop a secure car parking system based on image processing and implement on server side. The proposed application will protect the security of parking area. This application provides an efficient platform for tracking unauthorized car using matlab extract number of car and send to it on server with notification to the monitor of the entire system.

The main reason for doing this project is solve the parking problem of our university. In University of Karachi there are limited gates and number of vehicles passing through each gate, which when checked manually raises inconvenience and also to depot five to six persons on each gate. This problem is aimed to solve through this computer program.

**1.3 Scope of the System.**

The main target users for our project would be anyone this project can be placed in universities, shopping malls, hospitals, airports and can be placed anywhere. Its UI is based on web interface so don’t need any kind of installation. Our main focus to design user experience very friendly so that user can use it easily. This system is based on fully automation user just start it on one time.

# **1.4 Project Objectives.**

The core objectives which have been designated as fundamental to the project are:

1. Registration/Un-registration of the car,
2. Detection of the car,
3. Pictures of the number plate,
4. Alarm to the monitoring department,
5. Identify all the vehicles that are registered in our database and make sure that any ill-legal vehicle cannot be parked in our premises.

# **1.5 Current System.**

There are many systems and lots of security companies’ provide different kind of security.

Some provides only extracted number and some provides only desktop version. In many others there is an issue of complex UI and one of the major issue that a person who monitor

this system can easily allowed to pass unauthorized car.

# **1.6 Proposed System.**

Our system how different from others there are following points.

1. Web base UI.
2. Highly secure.
3. If monitors of car parking area allow any unauthorized car then they are also catchable because we provides mail system which send mail to the head of all monitors when any unauthorized car enters.

.

# **1.7 Significance.**

On the individual level, security is most often understood as safety. This safety includes freedom from harm, whether physical or psychological. This system is an extremely secure which creates a strong security around a property or system. It is important in maintaining security level of a building or property parking. This is boosted by the use of technological security system. It makes management centralized and easy to monitor with just the tip of a finger.

**Benefits of access control system**

· Protection of the system from unauthorized use/entry. Only authorized persons can enter into parking area

· Reduces human resource costs as all the work is done in place of the physical workforce. In addition, there is no chance of losing money as it is more secure than if human staffs were used.

· Automatic controllers operations make the system well streamlined than in human management. It gives accurate measures and counts hence avoiding any errors.

# **1.8 Background**

We took an idea from different our society and searched some books how to implement this project. We consulted our teachers for the acceptance of this project and the way we will deliver the best performance. Other than this, we noticed there is a need to develop a car parking security system which will improve parking security of our university and also reduce the risk of any terrorism activity.

# **2 SDLC**

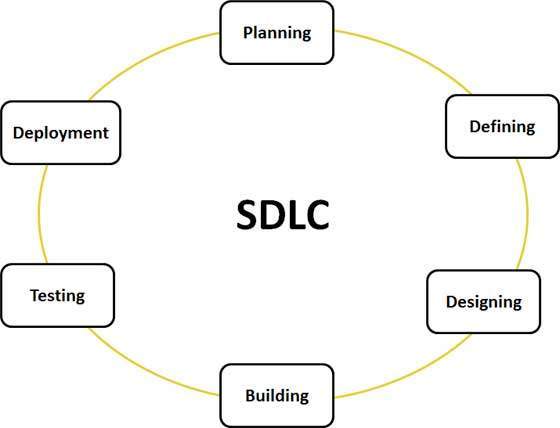
SDLC, Software Development Life Cycle is a process used by software industry to design, develop and test high quality software. The SDLC aims to produce a high quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

* SDLC is the acronym of Software Development Life Cycle.
* It is also called as Software development process.
* The software development life cycle (SDLC) is a framework defining tasks performed at each step in the software development process.
* ISO/IEC 12207 is an international standard for software life-cycle processes. It aims to be the standard that defines all the tasks required for developing and maintaining software.

## What is SDLC?

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development life cycle consists of the following stages

## Stage 1: Planning and Requirement Analysis

Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

## Stage 2: Defining Requirements

Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through .SRS. . Software Requirement Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

## Stage 3: Designing the product architecture

SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity , budget and time constraints , the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

## Stage 4: Building or Developing the Product

In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers have to follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers etc are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

## Stage 5: Testing the Product

This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However this stage refers to the testing only stage of the product where products defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

## Stage 6: Deployment in the Market and Maintenance

Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometime product deployment happens in stages as per the organizations. business strategy. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

There are various software development life cycle models defined and designed which are followed during software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type, in order to ensure success in process of software development. But in this project we select water fall model.

# **2.2 Waterfall Model.**

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Following is a diagrammatic representation of different phases of waterfall model.



The sequential phases in Waterfall model are:

* **Requirement Gathering and analysis:** **:** All possible requirements of the system to be developed are captured and documented in a requirement specification phase. This is the most crucial part of project because the whole project depends on what we required from this system. In this part, we made our document in which we write down all specifications of our project and full information about hardware/software which will be required for making this project.
* **SystemDesign:** After documenting all gathered information in requirement specifications phase, it’s time to prepare the design of the system. An important consideration while designing parking system has been taken for the use of appropriate technology for the detection of obstacles while capturing a vehicle. There are several technologies available in the market today to solve this problem. These include various types of radar, digital camera, infrared sensors, and ultrasound sensors. Each technology has its advantages and disadvantages and, after careful consideration of all available options, we decided to use infrared sensors in our design of parking system.
* **Implementation:**
* **Interface**: For implementing the design that we make earlier, I decided to program the entrance interface in **-----** mainly for its graphical environment and ease of use. Alternatively, the interface could have been programmed in JAVA, C++ or any other object oriented programming language.
* **Code**: An implementation can be done in many ways in which we decide to code it in units form. Then by taking an input from system design, the first phase were developed in small program called a unit, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
* **Database**: There are several databases are available in the market like mysql, sql, mangoes dB. We decide to go with mangoes because it has some additional features over sql and other database management system and mainly it is easy to use and for future modification in databases.
* **Integration and Testing:**

1. **Integration:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
2. **Testing**: Each component will be tested individually before integration.
   * Compile and run Matlab program
   * Testing the interface for all user input possibilities
   * Testing the connection database for all user input possibilities

Test integrated system as each feature is added. First, I will test assuming the ideal case (one car). Next, I will simulate and test the system’s capabilities for guiding multiple cars.

* **Deployment of system:** Once the functional and non-functional testing is done, the product is deployed in the customer environment or released into the market. It is the responsibility of deployment team to install and configure this system in customer’s pc and train them how to use this system.
* **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

# **3 Image Processing**

**3.1 Background**

Indexing images or videos requires information about their content. This content is often strongly related to the textual information appearing in them,which can be divided into two groups:

Text appearing accidentally in an image that usually does not represent anything important related to the content of the image. Such texts are referred to as scene text

Text produced separately from the image is in general a very good key to understand the image. In it is called artiﬁcial text.

In contrast to scene text, artiﬁcial text is not only an important source of information but also a signiﬁcant entity for indexing and retrieval purposes. Localization of text and simpliﬁcation of the background in images is the main

Objective of automatic text detection approaches. However, text localization in complex images is an intricate process due to the often bad quality of images, different backgrounds or different fonts, colors, sizes of texts appearing in them. In order to be successfully recognizable by an OCR system, an image having text must fulﬁll certain requirements, like a monochrome text and background where the background-to-text contrast should be high. In this paper, we present an approach that allows to detect, localize and extract texts from color images with complex backgrounds. The approach is targeted towards being robust with respect to different kinds of text appearances, including font size, color and language. To achieve this aim, the main focus of the proposed algorithm is centered on the recognition of the speciﬁc edge characteristics of characters. Based on the way how possible text areas are detected and localized, our method can be classiﬁed as a connected component based approach. It essentially works as follows: Color images are ﬁrst converted to grayscale images. An edge image is generated using a contrast segmentation algorithm, which in turn uses the contrast of the character contour pixels to their neighboring pixels. This is followed by the analysis of the horizontal projection of the edge image in order to localize the possible text areas. After applying several heuristics to enhance the resulting image created in the previous step, an output image is generated that shows the text appearing in the input image with a simpliﬁed background. These images are ready to be passed to an OCR system. The software is completely written in JAVA to be able to easily run the code in parallel on possibly heterogeneous networked computing platforms. The performance of our approach is illustrated by presenting experimental results for different sets of images.

**3.2 What is Image Processing?**

**Image processing** is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually **Image Processing** system includes treating images as two dimensional signals while applying already set signal processing methods to them.

It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps.

* + Importing the image with optical scanner or by digital photography.
  + Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns that are not to human eyes like satellite photographs.
  + Output is the last stage in which result can be altered image or report that is based on image analysis.

**Purpose of Image processing**

The purpose of image processing is divided into 5 groups. They are:

1.      Visualization - Observe the objects that are not visible.

2.      Image sharpening and restoration - To create a better image.

3.      Image retrieval - Seek for the image of interest.

4.      Measurement of pattern – Measures various objects in an image.

5.      Image Recognition – Distinguish the objects in an image.

**Types**

The two types of methods used for Image Processing are Analog andDigital Image Processing. Analog or visual techniques of image processing can be used for the hard copies like printouts and photographs. Image analysts use various fundamentals of interpretation while using these visual techniques. The image processing is not just confined to area that has to be studied but on knowledge of analyst. Association is another important tool in image processing through visual techniques. So analysts apply a combination of personal knowledge and collateral data to image processing.

Digital Processing techniques help in manipulation of the digital images by using computers. As raw data from imaging sensors from satellite platform contains deficiencies. To get over such flaws and to get originality of information, it has to undergo various phases of processing. The three general phases that all types of data have to undergo while using digital technique are Pre- processing, enhancement and display, information extraction.

# **3.3 Edge Detection**

**Edge detection** is the name for a set of mathematical methods which aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed *edges*. The same problem of finding discontinuities in 1D signals is known as step detection and the problem of finding signal discontinuities over time is known as change detection. Edge detection is a fundamental tool in image processing, machine vision and computer vision, particularly in the areas of feature detection and feature extraction

**3.4 The Proposed Text Localization Method**

In this section, the processing steps of the proposed text localization approach are presented. Our intention is to build an automatic text localization and extraction system which is able to accept different types of still images (or video frames) possibly with a complex background. The system design is based on the following assumptions: (a) the input to our system can be a grayscale or a color image;

(b) the current version can only detect texts with a horizontal alignment, and (c) texts that are smaller than a certain (small) font size will not be detected.

In contrast to many other text detection approaches, our complete implementation has been written in the JAVA programming language, which allows the code to be easily distributed and run in parallel on heterogeneous platforms connected via the Internet. This allows you to treat text localization as a scalable compute-intensive application of the Grid computing paradigm. The different steps of our approach are as follows.

**Step#1: Image Preprocessing.** If the image data is not represented in YUV color space, it is converted to this color space by means of an appropriate transformation. In contrast to the approaches presented in [1, 2, 8] our system only uses the luminance data (Y channel of YUV) during further processing. After that, luminance value thresholding is applied to spread luminance values throughout the image and increase the contrast between the possibly interesting regions and the rest of the image.

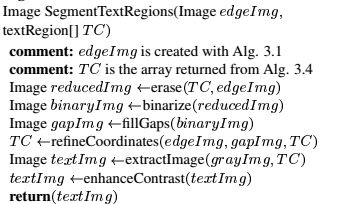
**Step#2: Edge Detection**. This step focuses the attention to areas where text may occur. We employ a simple method for converting the gray-level image into an edge image. Our algorithm (see Figure 1) is based on the fact that the character contours have high contrast to their local neighbors. As a result, all character pixels as well as some non-character pixels which also show high local color contrast are registered in the edge image. In this image, the value of each pixel of the original image is replaced by the largest difference between itself and its neighbors (in horizontal, vertical and diagonal direction). Despite its simplicity, this procedure is highly effective. Finally, the contrast between edges will be increased by means of a convolution with an appropriate mask.

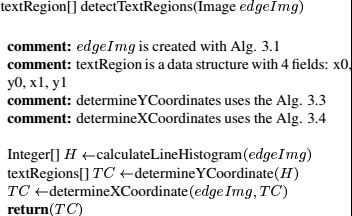
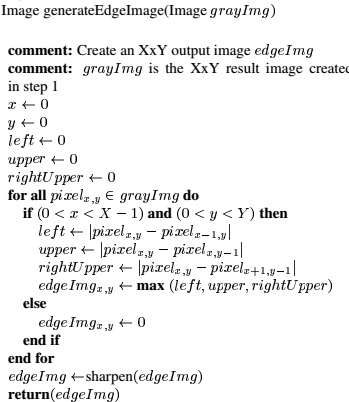
**Step#3: Detection of Text Regions.** The horizontal projection proﬁle of the edge image is analyzed in order to locate potential text areas. Since text regions show high contrast values, it is expected that they produce high peaks in horizontal projection. In subsequent processing, the local maxima are calculated by the histogram determined above. Two thresholds are employed to and the local maxima. A line of the image is accepted as a text line candidate if either it contains a sufficient number (Min-Edges) of sharp edges or the difference between the edge pixels in one line to its previous line is bigger than a threshold (Min-Line-Diff). Both thresholds are defined empirically and are fixed. In this way, a text region is isolated which may contain several texts aligned horizontal

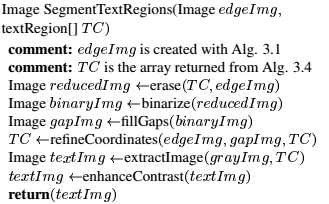
tally (whereby their y-coordinates are already defined). In a later step, we de ne the x-coordinates of the leftmost and rightmost, top and bottom point of the text region Finally, the exact coordinates for each of the detected areas are used to create bounding boxes.

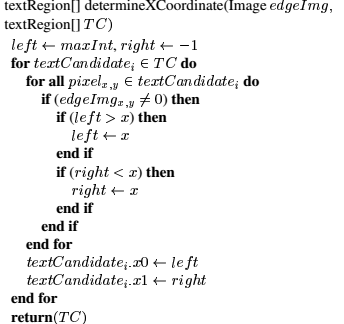
**Step #4: Enhancement and Segmentation of Text Regions.**

First, geometric properties of the text characters like the possible height, width, width to height ratio are used to discard those regions whose geometric features do not fall into the predefined ranges of values. All remaining text candidates undergo another treatment in order to generate the so- called text image where detected text appears on a simplified background. The binary edge image is generated from the edge image, erasing all pixels outside the predefined text boxes and then binarizing it. This is followed by the process of gap filling. If one white pixel on the binary edge image is surrounded by two black pixels in horizontal, vertical or diagonal direction, then it is also filled with black. The gap image is used as a reference image to re ne the localization of the detected text candidates. Text segmentation is the next step to take place. It starts with extraction of text candidates from the gray image. Then, the segmentation process concludes with a procedure which enhances text to background contrast on the text image









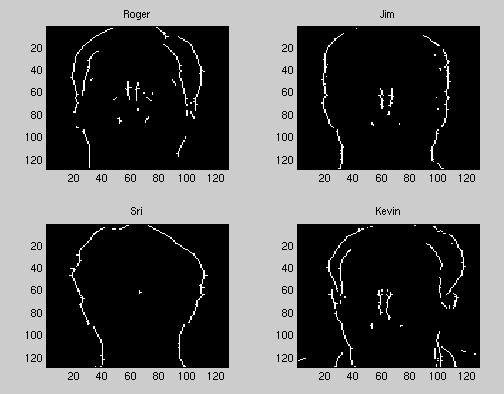
# **3.4 Other Methods for edge detection**

There are many ways to perform edge detection. However, the most may be grouped into two categories, gradient and Laplacian. The gradient method detects the edges by looking for the maximum and minimum in the first derivative of the image. The Laplacian method searches for zerocrossings in the second derivative of the image to find edges. This first figure shows the edges of an image detected using the gradient method (Roberts, Prewitt, Sobel) and the Laplacian method (Marrs-Hildreth).

|  |
| --- |
| http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/kfig1.jpg  **Various Edge Detection Filters** |

Notice that the facial features (eyes, nose, mouth) have very sharp edges. These also happen to be the best reference points for morphing between two images. Notice also that the Marr-Hildreth not only has a lot more noise than the other methods, the low-pass filtering it uses distorts the actual position of the facial features. Due to the nature of the Sobel and Prewitt filters we can select out only vertical and horizontal edges of the image as shown below. This is very useful since we do not want to morph a vertical edge in the initial image to a horizontal edge in the final image. This would cause a lot of warping in the transition image and thus a bad morph.

|  |
| --- |
| http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/kfig2.jpg  Vertical and Horizontal Edges |

The next pair of images show the horizontal and vertical edges selected out of the group members images with the Sobel method of edge detection. You will notice the difficulty it had with certain facial features, such as the hairline of Sri and Jim. This is essentially due to the lack of contrast between their hair and their foreheads.

|  |
| --- |
| http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/kfig8a.jpg Sobel Filtered Common Edges http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/kfig8b.jpg Sobel Filtered Common Edges |

We see that although it does do better for some features (ie. the nose), it still suffers from mis mapping some of the lines. A morph constructed using individually selected points would still work better. It should also be noted that this method suffers the same drawbacks as the previous page; difficulties due to large contrast between images and the inability to handle large translations of features. Another method of detecting edges is using wavelets. Specifically a two-dimensional Haar wavelet transform of the image produces essentially edge maps of the vertical, horizontal, and diagonal edges in an image.

This can be seen in the figure of the transform below, and the following figure where we have combined them to see the edges of the entire face.

And here are the maps of common control points generated by the feature extraction algorithm for the Jim-Roger morph.

|  |
| --- |
| http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/kfig9b.jpghttp://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/kfig9a.jpgHaar FilteredCommon Edges |

**4 Automate Process**

A general technology term that is used to describe any process being automated through the use of computers and computer software. Processes that have been automated require less human intervention and less human time to deliver

Software engineering is concerned with the development and evolution of large and complex software-intensive systems. It covers theories, methods and tools for the specification, architecture, design, testing, and maintenance of software systems. Today’s software systems are significantly large, complex and critical, that only through the use of automated approaches can such systems be developed and evolve in an economic and timely manner.

Automated software engineering applies computation to software engineering activities. The goal is to partially or fully automate these activities, thereby significantly increasing both quality and productivity. This includes the study of techniques for constructing, understanding, adapting and modelling both software artifacts and processes. Automatic and collaborative systems are both important areas of automated software engineering, as are computational models of human software engineering activities. Knowledge representations and artificial intelligence techniques applicable in this field are of particular interest, as are formal techniques that support or provide theoretical foundations.

Automated software engineering approaches have been applied in many areas of software engineering. These include requirements definition, specification, architecture, design and synthesis, implementation, modelling, testing and quality assurance, verification and validation, maintenance and evolution, configuration management, deployment, reengineering, reuse and visualisation. Automated software engineering techniques have also been used in a wide range of domains and application areas including industrial software, embedded and real-time systems, aerospace, automotive and medical systems, Web-based systems and computer games.

**4.1 Gulp**

* **Automation** - gulp is a toolkit that helps you automate painful or time-consuming tasks in your development workflow.
* **Platform-agnostic** - Integrations are built into all major IDEs and people are using gulp with PHP, .NET, Node.js, Java, and other platforms.
* **Strong Ecosystem** - Use npm modules to do anything you want + over 2000 curated plugins for streaming file transformations
* **Simple** - By providing only a minimal API surface, gulp is easy to learn and simple to use
  1. **How to use Gulp?**

**1. Install gulp globally:**

**If you have previously installed a version of gulp globally, please run npm rm --global gulp to make sure your old version doesn't collide with gulp-cli.**

$ npm install --global gulp-cli

**2. Install gulp in your project devDependencies:**

$ npm install --save-dev gulp

**3. Create a gulpfile.js at the root of your project:**

var gulp = require('gulp');

gulp.task('default', function() {

// place code for your default task here});

**4. Run gulp:**

$ gulp The default task will run and do nothing. To run individual tasks, use gulp <task> <othertask>.

**4.3 Why we use task runner?**

In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you've configured it through a [Gulpfile](http://gruntjs.com/sample-gruntfile), a task runner can do most of that mundane work for you—and your team—with basically zero effort.

**4.4 Why use Gulp?**

The Gulp ecosystem is huge and it's growing every day. With literally hundreds of plugins to choose from, you can use Gulp to automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Gulp plugin to npm is a breeze

# **5 Project Overview**

# 5.1 Registration panel

# 5.2 Logs

# 5.3 Entries

# 5.4 Live Streaming

# 5.5 Notifications

# 5.6 Task runner

# 5.7 Database

Gulp is continuously watching notepad file and restarts server

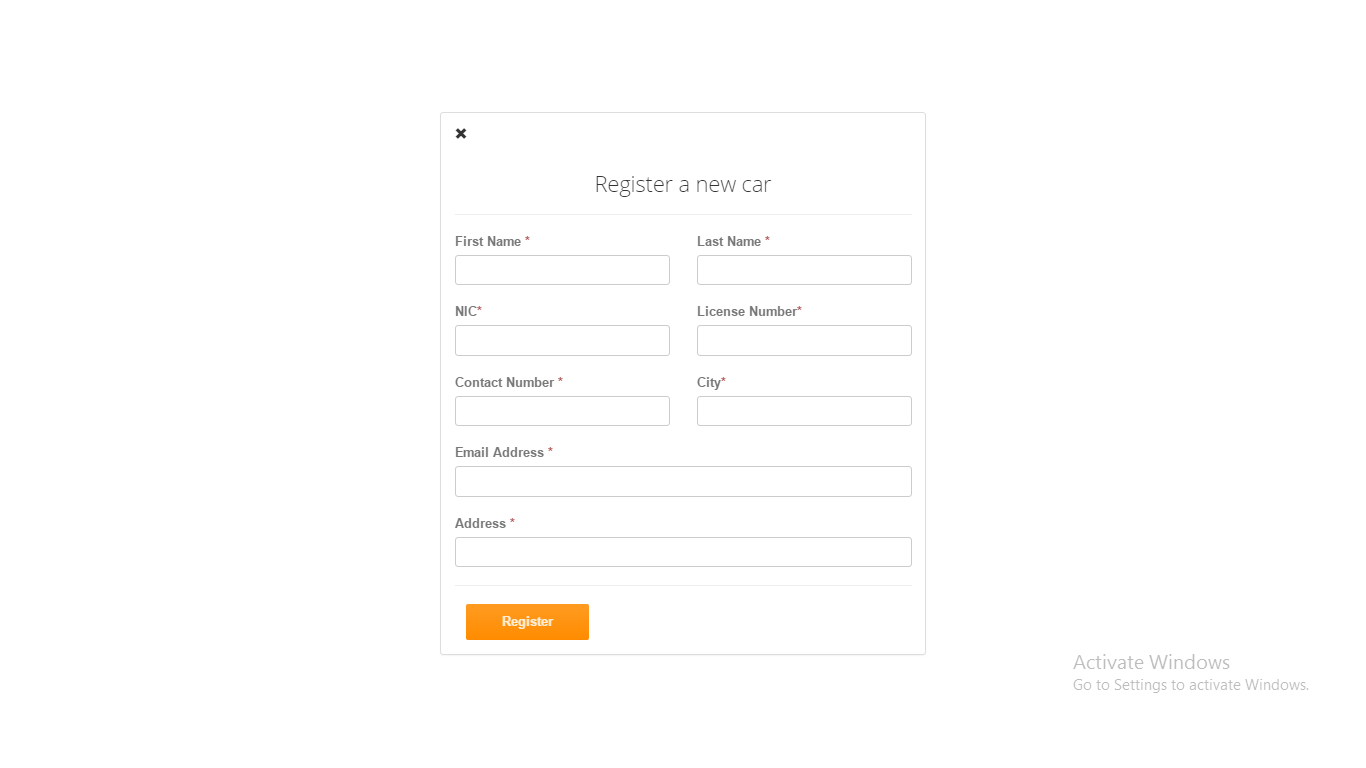
Matlab is extracting number plate of car

Notepad file

Nodejs server reads changes from notepad file and stores in db

Database

# **5.1 Registration Panel**



This is the registration panel it used to register new entry of car. All fields are necessary to fill except email. Every entry’s detail sends to that person who monitors this complete system.

**CodeSnippet**

<form action="/" method="POST">

<div class="row top-margin">

<div class="col-sm-6">

<label>First Name <span class="text-danger">\*</span></label>

<input type="text" class="form-control" name="firstname">

</div>

<div class="col-sm-6"><label>Last Name <span class="text-danger">\*</span></label>

<input type="text" class="form-control" name="lastname">

</div>

</div>

<div class="row top-margin">

<div class="col-sm-6">

<label>NIC<span class="text-danger">\*</span></label>

<input type="number" class="form-control" name="nic">

</div>

<div class="col-sm-6">

<label>License Number<span class="text-danger">\*</span></label>

<input type="text" class="form-control" name="licensenumber">

</div>

</div>

<div class="row top-margin">

<div class="col-sm-6">

<label>Contact Number <span class="text-danger">\*</span></label>

<input type="number" class="form-control" name="contactnumber">

</div>

<div class="col-sm-6">

<label>City<span class="text-danger">\*</span></label>

<input type="text" class="form-control" name="city">

</div>

</div>

<div class="top-margin">

<label>Email Address <span class="text-danger">\*</span></label>

<input type="email" class="form-control" name="email">

</div>

<div class="top-margin">

<label>Address <span class="text-danger">\*</span></label>

<input type="text" class="form-control" name="address">

</div>

<hr>

<div class="row"><div class="col-lg-4 text-right">

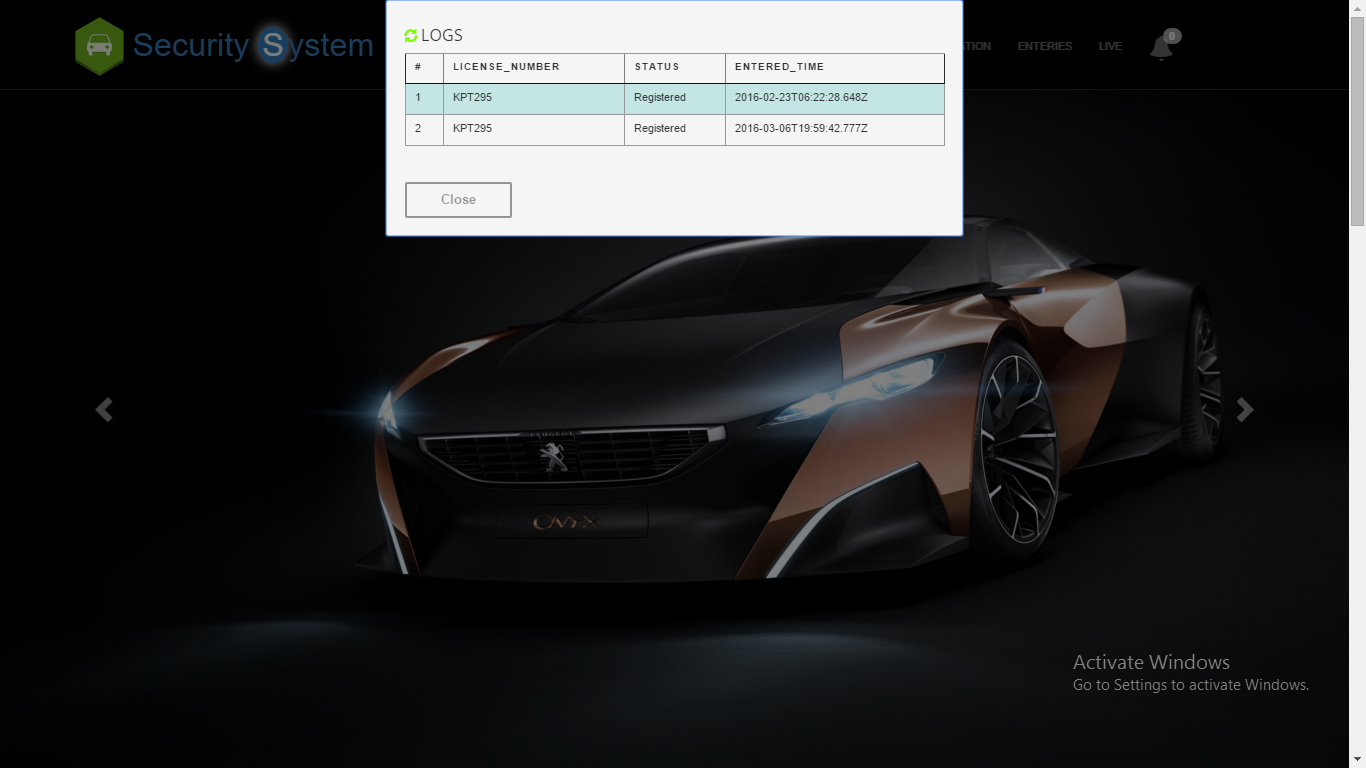
<button class="btn btn-action" type="submit">Register</button>

</div>

</div>

</form>

# **5.2 Logs**



This area shows daily entries of cars whether it is register or unregister and these logs are clear automatically after one day.

**CodeSnippet**

<div id="slide" class="well">

<span id="refresh" onclick="ajaxCall()">

<i class="glyphicon glyphicon-refresh"></i>

<h4>LOGS</h4><br/></span>

<table id="hor-minimalist-a" summary="Employee Pay Sheet" class="scroll prettyprint" border="1">

<thead>

<tr>

<th scope="col">#</th>

<th scope="col">License\_Number</th>

<th scope="col">Status</th>

<th scope="col">Entered\_Time</th>

</tr>

</thead>

<tbody id='row-data'>

</tbody>

</table>

<br><br>

<button class="slide\_close btn btn-default">Close</button>

<script>

$(document).ready(function () {

$('#slide').popup({

focusdelay: 400,

outline: true,

vertical: 'top'

});

});

var logs= <%- logs %>

var count=[];

for(var i=0;i<logs.length;i++){

if(logs[i].status=="Unregistered"){

count.push(logs[i].status);

}

}

$('#count').append(count.length);

if(logs==" "){}

else{

for(var i=0;i<logs.length;i++){

$('#row-data').append(

"<tr class='even' >"

+"<td>"+(i+1)+"</td>"

+"<td>"+logs[i].plateNumber+"</td>"

+"<td class='logs'>"+logs[i].status+"</td>"

+"<td>"+logs[i].createdOn+"</td>"

+"</tr>")

}

console.log(logs);

}

//setInterval(ajaxCall, 3000); //300000 MS == 5 minutes

function dummy(){console("ok");

}

function ajaxCall() {

document.getElementById('row-data').innerHTML=" "

document.getElementById('count').innerHTML=" "

var data={"number":"AXZ-420","EnteringTime":Date()}

$.ajax({

type: 'GET',

data: data,

cache: false,

contentType: 'application/json',

datatype: "json",

url: '/fetch',

success: function (logs) {

var \_data=JSON.parse(logs);

// console.log( \_data);

//$.notify(\_data,"warning");

for(var i=0;i<\_data.length;i++){

$('#row-data').append(

"<tr class='even' >"

+"<td >"+(i+1)+"</td>"

+"<td >"+\_data[i].plateNumber+"</td>"

+"<td class='logs'>"+\_data[i].status+"</td>"

+"<td >"+\_data[i].createdOn+"</td>"+"</tr>")}

var count=[];

for(var i=0;i<\_data.length;i++){

if(\_data[i].status=="Unregistered"){

count.push(\_data[i].status);

}

}

$('#count').append(count.length);

console.log($('#count').text())

if(($('#count').text())>0){

$('#count').css('background','red');

var audio = document.getElementById("audio");

audio.play();

//alert("Hey..... Unauthorized car has detected take some action", function() { });

} } });}

if(($('#count').text())==0){

$('#count').css('background','#9d9d9d');

}

if(($('#count').text())>0){

var audio = document.getElementById("audio");

//audio.play();

bootbox.alert("Hey..... Unauthorized car has detected take some action", function() { });

}

setInterval(warn,100000);

function warn(){

if(($('#count').text())>0){

$('#count').css('background','red');

var audio = document.getElementById("audio");

audio.play();

bootbox.alert("Hey..... Unauthorized car has detected take some an action", function() { });

}

}

var col=$(".logs");

for(var i=0 ;i<col.length; i++)

{

if(col[i].innerText=='Unregistered'){

var \_i = document.createElement("i");

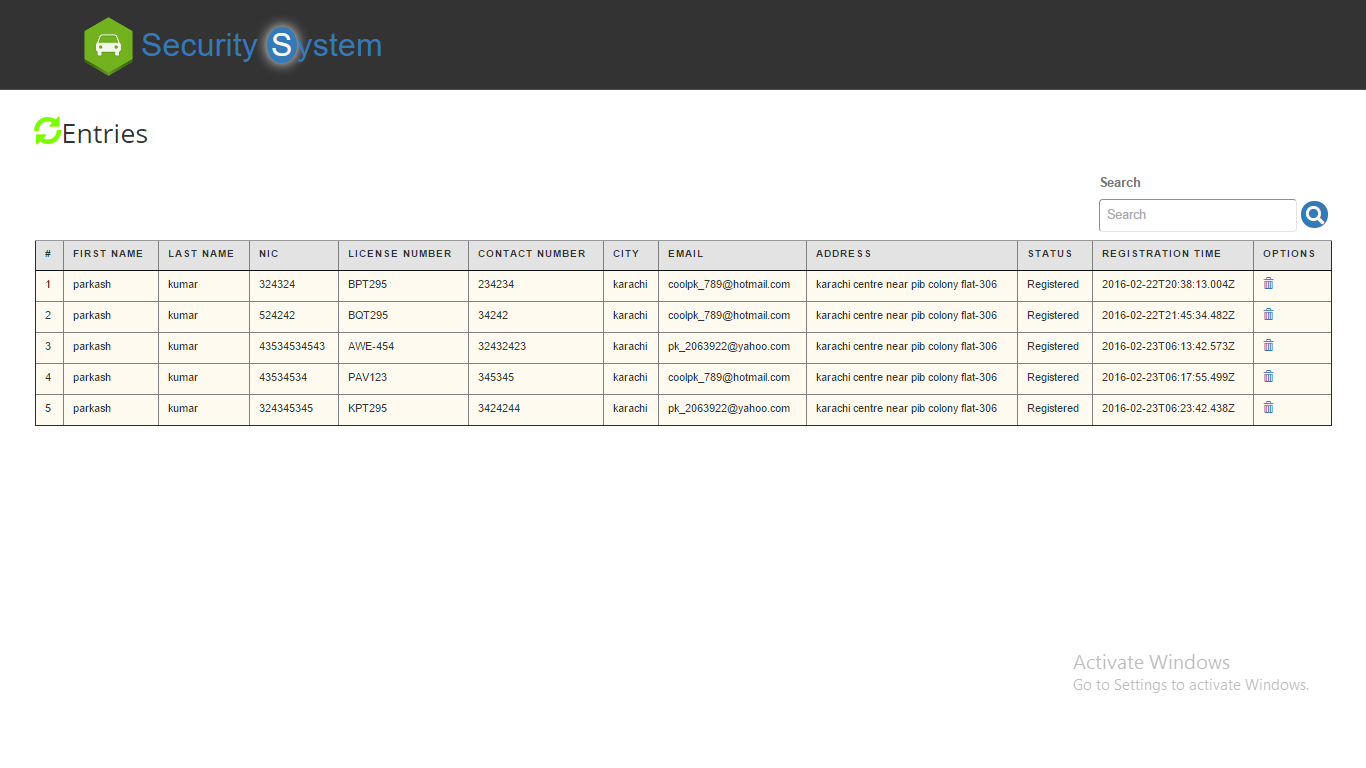
col[i].appendChild(\_i);

//console.log(col[i]);}}

</script>

</script>

# **5.3 Entries**



This area shows complete details of register car with some options i.e search and delete user can search specific record and also delete any record

**CodeSnippet**

<div id="fade" class="well1">

<i class='fade\_close glyphicon glyphicon-remove' ></i>

<span id="refresh" onclick="enteriesAjaxCall()">

<i class="glyphicon glyphicon-refresh"></i>

<h4>Entries</h4><br/><span>

<table id="hor-minimalist-a" summary="Employee Pay Sheet" class="scroll prettyprint entriestable" border="1" >

<thead class="entry-head">

<tr>

<th scope="col">#</th>

<th scope="col">First Name</th>

<th scope="col">Last Name</th>

<th scope="col">NIC</th>

<th scope="col">License Number</th>

<th scope="col">Contact Number</th>

<th scope="col">City</th>

<th scope="col">Email</th>

<th scope="col">Address</th>

<th scope="col">Status</th>

<th scope="col">Registration Time</th>

<th scope="col">Options</th>

</tr>

</thead>

<tbody id='row-data-enteries' class="entry-body">

</tbody>

</table>

</div>

<script>

$(document).ready(function () {

$('#fade').popup({

transition: 'all 0.3s',

scrolllock: true

});

});

function enteriesAjaxCall() {

document.getElementById('row-data-enteries').innerHTML=" "

document.getElementById('count').innerHTML=" "

var data={"number":"AXZ-420","EnteringTime":Date()}

$.ajax({

type: 'GET',

data: data,

cache: false,

contentType: 'application/json',

datatype: "json",

url: '/enteries',

success: function (logs) {

var \_data=JSON.parse(logs);

console.log( \_data);

//$.notify(\_data,"warning");

for(var i=0;i<\_data.length;i++){

if(\_data!=" "){

$('#row-data-enteries').append(

"<tr class='even' >"

+"<td>"+(i+1)+"</td>"

+"<td>"+\_data[i].firstname+"</td>"

+"<td>"+\_data[i].lastname+"</td>"

+"<td>"+\_data[i].nic+"</td>"

+"<td>"+\_data[i].licenseNumber+"</td>"

+"<td>"+\_data[i].contactNumber+"</td>"

+"<td>"+\_data[i].city+"</td>"

+"<td>"+\_data[i].email+"</td>"

+"<td>"+\_data[i].address+"</td>"

+"<td>"+\_data[i].status+"</td>"

+"<td>"+\_data[i].createdOn+"</td>"

+"<td><a href='' id="+\_data[i].\_id+" onclick='deleteAjaxCall(id)'><i class='glyphicon glyphicon-trash'></i></a></td>"

+"</tr>")

}

} }

});}

function deleteAjaxCall(id) {

var data={"id":id};

console.log(data);

$.ajax({

type: 'GET',

data: data,

cache: false,

contentType: 'application/json',

datatype: "json",

url: '/delete',

success: function (logs) {

var \_data=JSON.parse(logs);

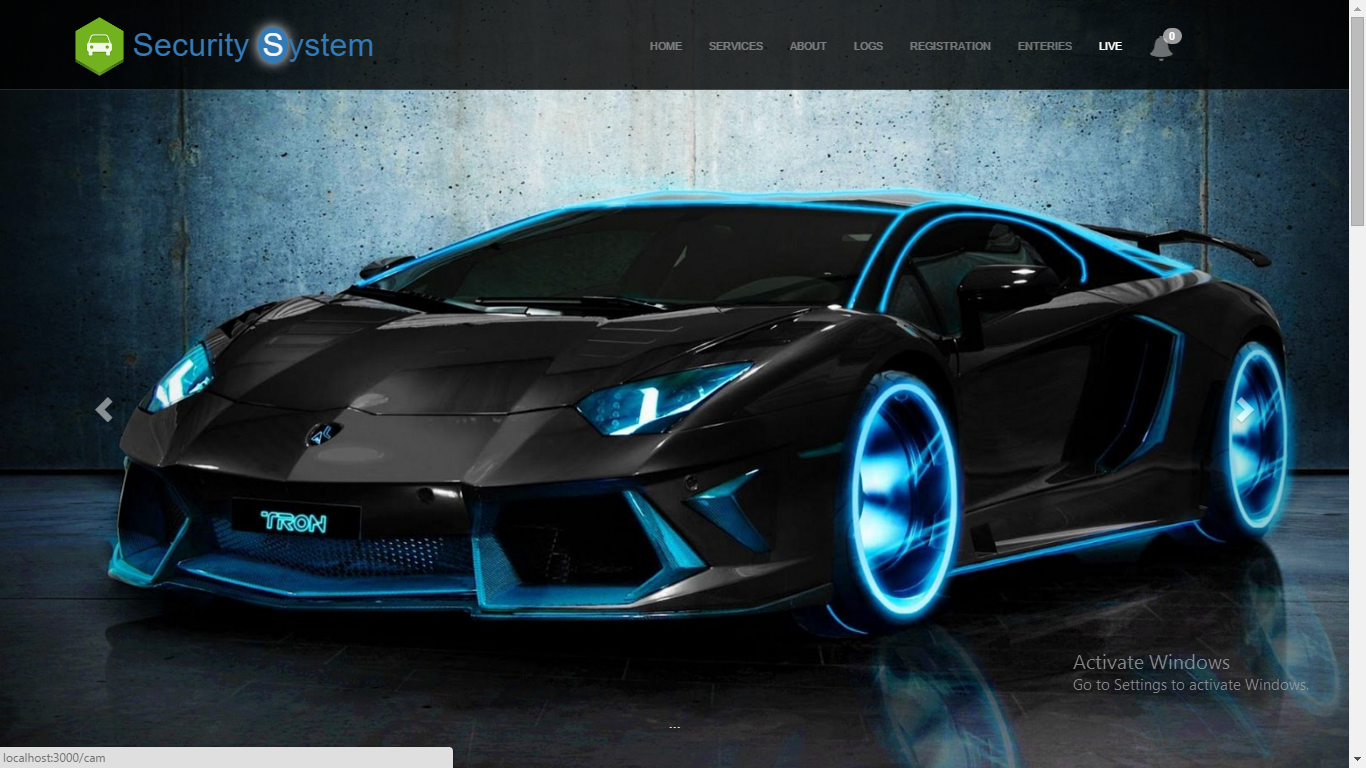
bootbox.alert("Record Deleted", function() { });

}});

}

</script>

# **5.4 Live Streaming**



With the help of live streaming any member of security can see the entries of cars

**CodeSnippet**

<!Doctype html>

<html>

<head>

<title>

</title>

<style>

#canvas {

margin-top: 20px;

border: 1px solid #ccc;

display: inline-block;

margin-right:4em;

}

</style>

</head>

<body>

<video id="video" width="640" height="480" autoplay></video>

<canvas id="canvas" width="640" height="480" x="640"></canvas><br/>

<button id="snap">Snap Photo</button>

</body>

<script>

// Put event listeners into place

window.addEventListener("DOMContentLoaded", function() {

// Grab elements, create settings, etc.

var canvas = document.getElementById("canvas"),

context = canvas.getContext("2d"),

video = document.getElementById("video"),

videoObj = { "video": true },

errBack = function(error) {

console.log("Video capture error: ", error.code);

};

// Put video listeners into place

if(navigator.getUserMedia) { // Standard

navigator.getUserMedia(videoObj, function(stream) {

video.src = stream;

video.play();

}, errBack);

} else if(navigator.webkitGetUserMedia) { // WebKit-prefixed

navigator.webkitGetUserMedia(videoObj, function(stream){

video.src = window.URL.createObjectURL(stream);

video.play();

}, errBack);

}

else if(navigator.mozGetUserMedia) { // Firefox-prefixed

navigator.mozGetUserMedia(videoObj, function(stream){

video.src = window.URL.createObjectURL(stream);

video.play();

}, errBack);

}

document.getElementById("snap").addEventListener("click", function() {

context.drawImage(video, 0, 0, 640, 480);

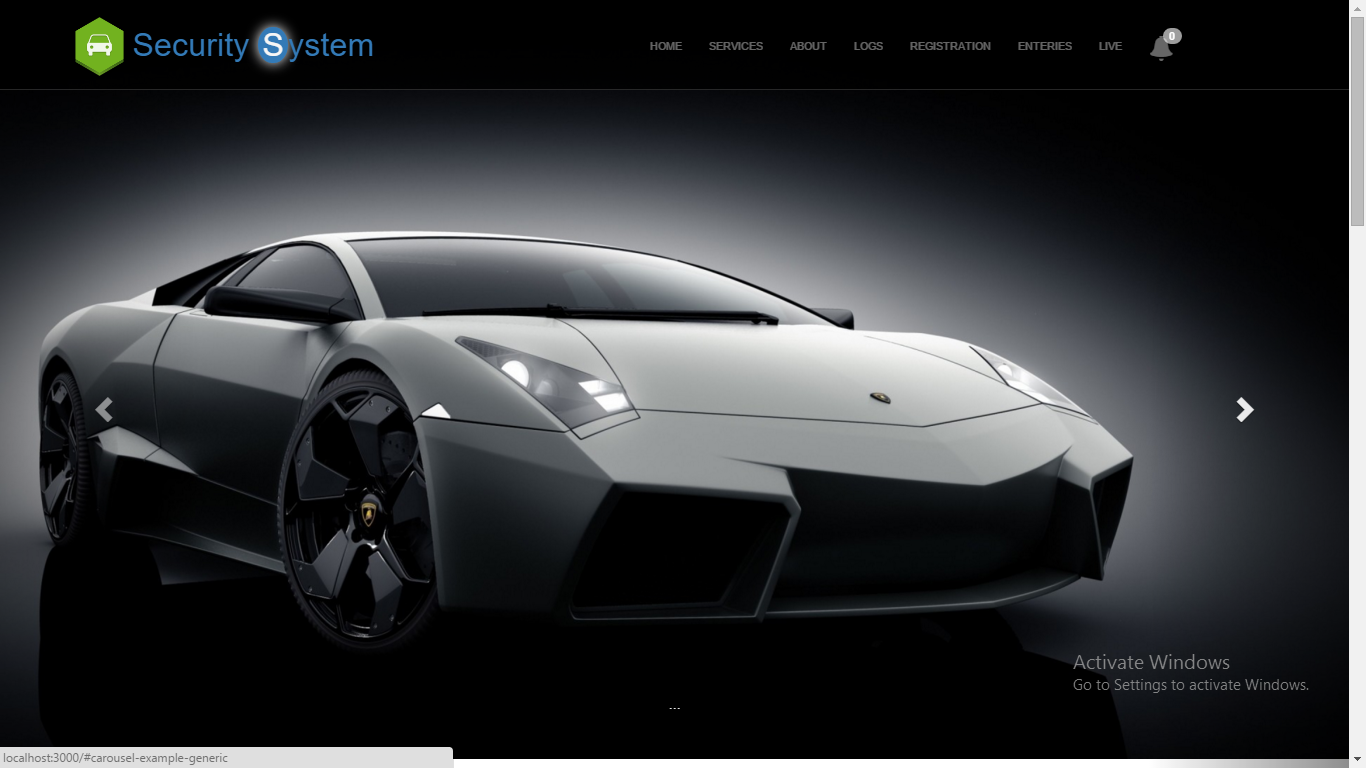
});

}, false);

</script>

</html>

# **5.5 Notifications**



This shows the number of unregistered car entered in the parking area by default its color is grey but when unregistered car is detected it turns into red.

**CodeSnippet**

<li><a class="font-horn" ><i class="glyphicon glyphicon-bell" data-toggle="tooltip" data-placement="bottom" title="Unregistered Entries"></i><span class="badge" id="count"></span></a></li>

var count=[];

for(var i=0;i<\_data.length;i++){

if(\_data[i].status=="Unregistered"){

count.push(\_data[i].status);

}

}

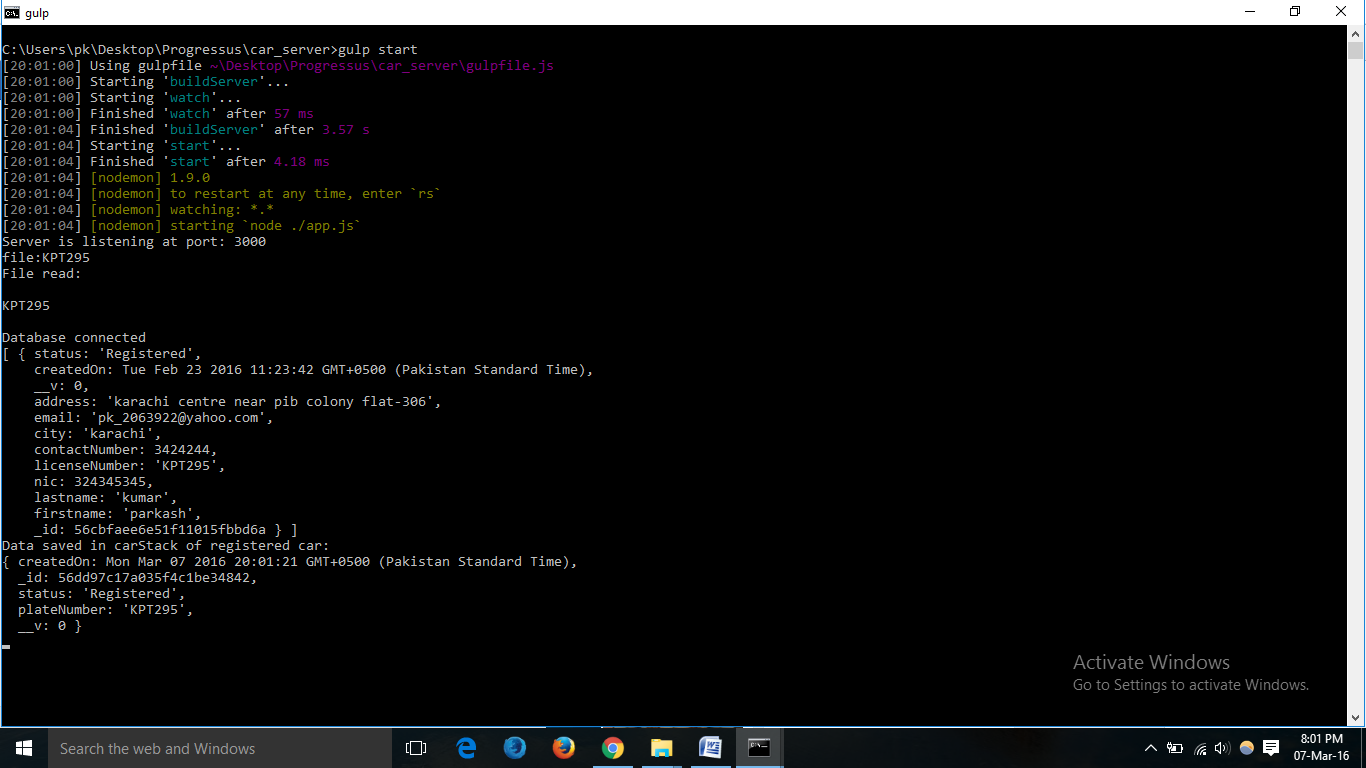
$('#count').append(count.length);

//console.log(data);

console.log($('#count').text())

if(($('#count').text())>0){$('#count').css('background','red');

# **5.6 Task Runner**



Gulp is used as a task runner which automatically restarts server and updates database. It is continously watching notepad file where extracted number is stored.

**CodeSnippet**

var gulp = require('gulp');

var ts = require('gulp-typescript');

var rimraf = require('gulp-rimraf');

var nodemon = require('gulp-nodemon');

gulp.task('buildServer', function () {

var tsResult = gulp.src('./app.ts')

.pipe(ts({

module: 'CommonJS'

}));

return tsResult.js.pipe(gulp.dest('./'));

});

gulp.task('start', ['buildServer', 'watch'], function(){

nodemon({

script: './app.js',

text:'./numberPlate.txt'

}).on('restart', function(){

console.log('nodemon restarted server.js');

})

})

gulp.task('watch', function() {

gulp.watch(['./app.ts','./numberPlate.txt'], ['buildServer']);

});

gulp.task('default', ['buildServer']);

gulp.task('open', function(){

var options = {

uri: 'localhost:3000',

app: 'chrome'

};

gulp.src('./app.js')

.pipe(open(options));

});

gulp.task('browser', ['start'], function() {

browserSync.init(null, {

proxy: "http://localhost:3000",

files: ["./\*.txt"],

browser: "google chrome",

port: 5000,

});

});

gulp.task('open', function(){

var options = {

uri: 'localhost:3000',

app: 'chrome'

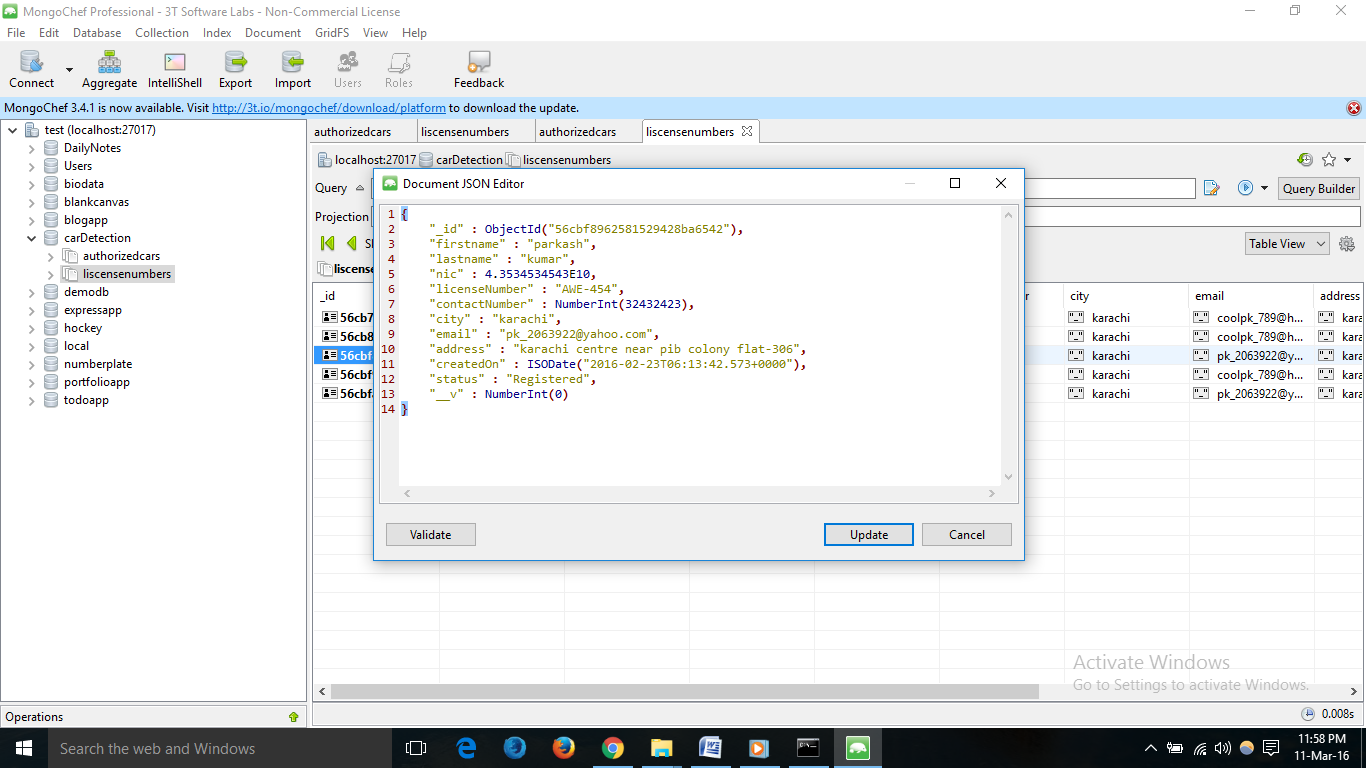
};

gulp.src('./app.js')

.pipe(open(options));

});

# **5.7 Database**

****

Mongodb stores data in JSON format and this is the json of document of registered car.

**CodeSnippet**

var mongoose=require('mongoose');

//connection string

var dburi="mongodb://127.0.0.1/carDetection";

mongoose.connect(dburi);

//verifying connection

mongoose.connection.on('connected',function () {

console.log("Database connected");

});

mongoose.connection.on('error',function (err) {

console.log("Error in db connection:\n"+err);

});

//Database schema

var car=mongoose.Schema({

"firstname":{type:String,required:true},

"lastname":{type:String,required:true},

"nic":{type:Number,required:true,unique:true},

"licenseNumber":{type:String,required:true,unique:true},

"contactNumber":{type:Number,required:true},

"city":{type:String,required:true},

"email":{type:String},

"address":{type:String,required:true},

"status":{type:String,default:"Registered"},

"createdOn":{type:Date,default:Date.now()}

})

var trackCar=mongoose.Schema({

"plateNumber":{type:String},

"status":{type:String },

"createdOn":{type:Date,default:Date.now()} ,})

//Database Model

var liscenseNumber=mongoose.model('liscenseNumber',car);

var AuthorizedCar=mongoose.model('AuthorizedCar',trackCar);

var carStack=new AuthorizedCar({

"plateNumber":num,

"status":"Unregistered",

"createdOn":Date.now() ,

}).save(function(err,data){

var result=data;

if(err)console.log("Error to maintain data of unregistered car: \n"+err);

else{

console.log("Data saved in carStack of unregistered car: \n"+data);

var mailOptions = {

from: "RedZone Security Systems pkbscs67@gmail.com", // sender address

to: "pk\_bscs@yahoo.com", // list of receivers

subject: "Alert...Unregistered car detected", // Subject line

text:"Unregistered car of number is: "+data.plateNumber

// plaintext body

// html body}

// send mail with defined transport object

// send mail with defined transport object

transporter.sendMail(mailOptions, function(error, info){

if(error){

return console.log(error);}

console.log('Message sent: ' + info.response);});}})}

app.post('/',function (req,res) {

var car\_entry=new liscenseNumber({

"firstname":req.body.firstname,

"lastname":req.body.lastname,

"nic":req.body.nic,

"licenseNumber":req.body.licensenumber,

"contactNumber":req.body.contactnumber,

"city":req.body.city,

"email":req.body.email,

"address":req.body.address,

"status":"Registered",

"createdOn":Date.now()

}).save(function(err,data){

if(err)console.log("Data is not inserted: \n"+err);

else{

console.log("Data inserted: \n"+data);

var result=JSON.stringify(data);

var mailOptions = {

from: "RedZone Security Systems pkbscs67@gmail.com", // sender address

to: "pk\_bscs@yahoo.com", // list of receivers

subject: "New Registration", // Subject line

text:"firstname: "+req.body.firstname+"\n"

+"lastname: "+req.body.lastname+"\n"

+"NIC: "+req.body.nic+"\n"

+"licenseNumber: "+req.body.licensenumber+"\n"

+"contactNumber: "+req.body.contactnumber+"\n"

+"city: "+req.body.city+"\n"

+"email: "+req.body.email+"\n"

+"address: "+req.body.address+"\n" }

transporter.sendMail(mailOptions, function(error, info){

if(error){

return console.log(error);

}console.log('Message sent: ' + info.response);});

res.redirect('/'); }})});

var carStack=new AuthorizedCar({

"plateNumber":data[i].licenseNumber,

"status":data[i].status,

"createdOn":Date.now() ,

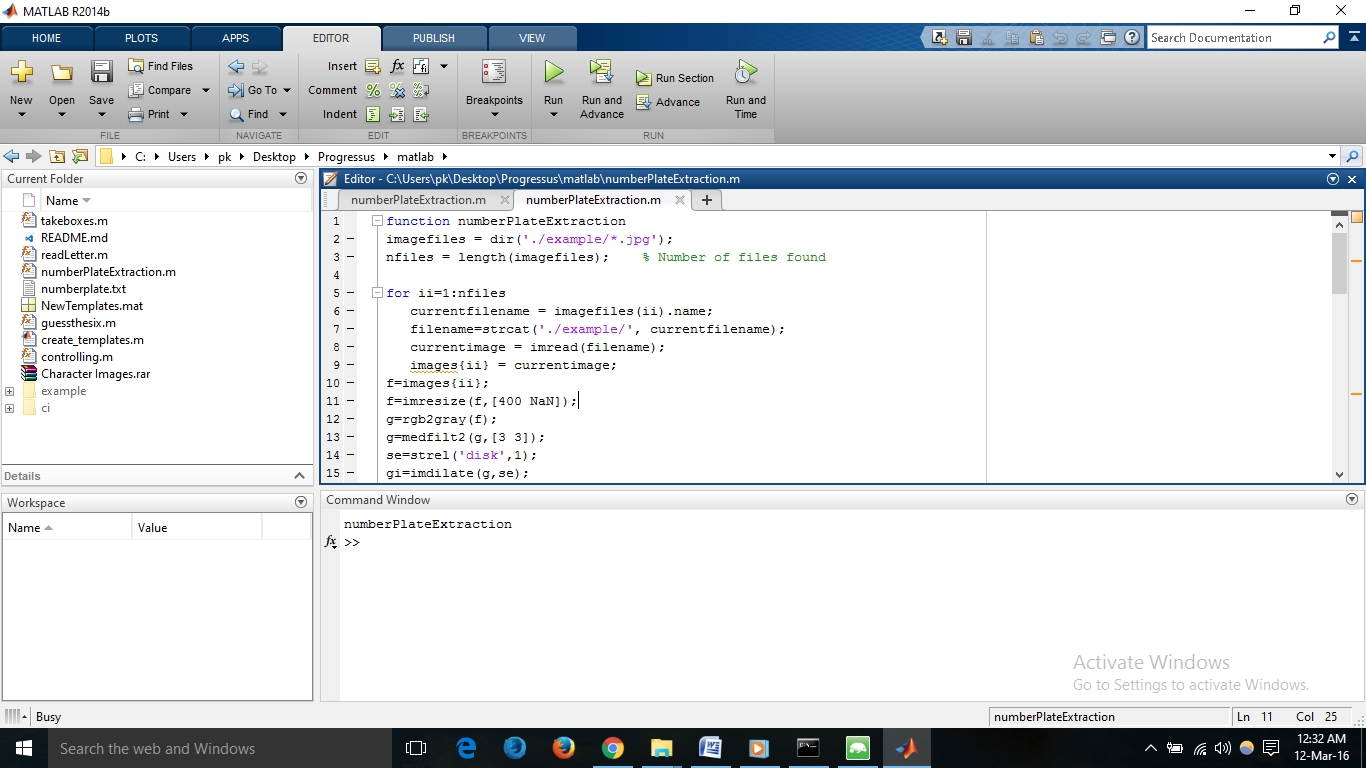
}).save(function(err,data){

if(err)console.log("Error to maintain data of registered car: \n"+err);

else{

console.log("Data saved in carStack of registered car: \n"+data);}})

# **5.8 Image Processing Using Matlab**

****

**CodeSnippet**

function numberPlateExtraction

imagefiles = dir('./example/\*.jpg');

nfiles = length(imagefiles); % Number of files found

for ii=1:nfiles

currentfilename = imagefiles(ii).name;

filename=strcat('./example/', currentfilename);

currentimage = imread(filename);

images{ii} = currentimage;

f=images{ii};

f=imresize(f,[400 NaN]);

g=rgb2gray(f);

g=medfilt2(g,[3 3]);

se=strel('disk',1);

gi=imdilate(g,se);

ge=imerode(g,se);

gdiff=imsubtract(gi,ge);

gdiff=mat2gray(gdiff);

gdiff=conv2(gdiff,[1 1;1 1]);

gdiff=imadjust(gdiff,[0.5 0.7],[0 1],0.1);

B=logical(gdiff);

er=imerode(B,strel('line',50,0));

out1=imsubtract(B,er);

F=imfill(out1,'holes');

H=bwmorph(F,'thin',1);

H=imerode(H,strel('line',3,90));

final=bwareaopen(H,100);

Iprops=regionprops(final,'BoundingBox','Image');

NR=cat(1,Iprops.BoundingBox);

r=controlling(NR);

if ~isempty(r)

I={Iprops.Image};

noPlate=[];

for v=1:length(r)

N=I{1,r(v)};

letter=readLetter(N);

while letter=='O' || letter=='0'

if v<=3

letter='O';

else

letter='0';

end

break;

end

noPlate=[noPlate letter];

end

fid = fopen('../car\_server/numberPlate.txt','w');

fprintf(fid,'\n%s \n\r',noPlate);

fclose(fid);

winopen('../car\_server/numberPlate.txt')

else

fprintf('Unable to extract the characters from the number plate.\n');

fprintf('The characters on the number plate might not be clear or touching with each other or boundries.\n');

end

pause(10);

end

end

# **6 Description of Hardware System**

# **6.1 CCTV / IP / IR Camera**

IP camera is used to capture image of car so that we number can be extracted number plate of car.

IP camera, is a type of digital video camera commonly employed for surveillance, and which, unlike analog closed circuit television (CCTV) cameras, can send and receive data via a computer network and the Internet. Although most cameras that do this are webcams, the term "IP camera" or "netcam" is usually applied only to those used for surveillance.

# **6.2 Computer or Laptop**

A computer or laptop is required to monitor or run the enitre security system. The hardware requirements which you are need to install this software.

|  |  |
| --- | --- |
| Operating System | Windows 7 + |
| Ram | 1GB |
| Space | 512mb |

# **7 Description of Technologies.**

# **7.1 What is Matlab?**

In [computer programming](https://en.wikipedia.org/wiki/Computer_programming), **Matlab** is an [integrated development environment](https://en.wikipedia.org/wiki/Integrated_development_environment) (IDE). It contains a base [workspace](https://en.wikipedia.org/wiki/Workspace) and an extensible [plug-in](https://en.wikipedia.org/wiki/Plug-in_(computing)) system for customizing the environment. It developed a proprietary programming language that allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python.

# **Interface**

MATLAB is a [multi-paradigm](https://en.wikipedia.org/wiki/Multi-paradigm_programming_language) [numerical computing](https://en.wikipedia.org/wiki/Numerical_analysis) environment and [fourth-generation programming language](https://en.wikipedia.org/wiki/Fourth-generation_programming_language). It supports developing applications with [graphical user interface](https://en.wikipedia.org/wiki/Graphical_user_interface) (GUI) features.GUI provide point-and-click control of software applications, eliminating the need to learn a language or type commands in order to run the application. MATLAB apps are self-contained MATLAB programs with GUI front ends that automate a task or calculation. The GUI typically contains controls such as menus, toolbars, buttons, and sliders. Many MATLAB products, such as Curve Fitting Toolbox, Signal Processing Toolbox, and Control System Toolbox, include apps with custom user interfaces. You can also create your own custom apps, including their corresponding UIs, for others to use.

# **Application workflow**

This application is basically divided into parts client-side and server-side. Client side consist on matlab and web-based user interface whereas on server side requests are handle by nodejs sever.

1. Gulp: It is task runner which is used to start nodejs server and continously watching any change in notepad file which stores number extracted from number plate of a car. If any change occurs in notepad file then it automatically restarts our server.

1. Matlab: It is used to extract number from a number plate of car and stores in a notepad file.
2. Nodejs: It is used to handle requests and extract number from a notepad file and stores in a database
3. Mongodb: It is used to store details of a car and daily entries.
4. Logs: It is an area in web where user can see all daily entries.
5. Ajax: Whenever new car is entered number is extracted in a notepad file through matlab and server gets restart with the help of gulp when server restarts an ajax request is triggered to nodejs server that checks the entered car number in a database if it is not present then marked is as an unregistered car.
6. Nodemailer: Whenever new license number is detect a mail is sent to a user with complete details of car

Gulp is continuously watching notepad file and restarts server

Matlab is extracting number plate of car

Notepad file

Nodejs server reads changes from notepad file and stores in db

Database

# **7.2 What is Nodejs?**

Node.js is an [open-source](https://en.wikipedia.org/wiki/Open-source_software), [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) [runtime environment](https://en.wikipedia.org/wiki/Runtime_system) for developing [server-side](https://en.wikipedia.org/wiki/Server-side) [Web applications](https://en.wikipedia.org/wiki/Web_application). Node.js is a platform built on Chrome’s JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices. It’s not a [JavaScript framework](https://en.wikipedia.org/wiki/JavaScript_framework) but many of its basic modules are written in [JavaScript](https://en.wikipedia.org/wiki/JavaScript), and developers can write new modules in JavaScript. The runtime environment interprets JavaScript using [Google](https://en.wikipedia.org/wiki/Google)'s [V8](https://en.wikipedia.org/wiki/V8_(JavaScript_engine)) JavaScript engine.

**Built on JavaScript**

For better or worse, JavaScript is the world’s most popular programming language.

If you’ve done any programming for the web, it’s unavoidable. JavaScript, because of the

sheer reach of the web, has fulfilled the “write once, run anywhere” dream that Java

had back in the 1990s.

Around the time of the Ajax revolution in 2005, JavaScript went from being a “toy”

language to something people wrote real and significant programs with. Some of the

notable firsts were Google Maps and Gmail, but today there are a host of web applications from Twitter to Facebook to GitHub.

Since the release of Google Chrome in late 2008, JavaScript performance has

improved at an incredibly fast rate due to heavy competition between browser vendors

(Mozilla, Microsoft, Apple, Opera, and Google). The performance of these modern

JavaScript virtual machines is literally changing the types of applications you can build

on the web. A PC emulator running in JavaScript where you can load a Linux kernel, interact with the terminal session, and compile a C program, all in your browser.

Node uses V8, the virtual machine that powers Google Chrome, for server-side programming. V8gives Node a huge boost in performance because it cuts out the middleman, preferring straight compilation into native machine code over executing

bytecode or using an interpreter. Because Node uses JavaScript on the server, there are also other benefits:

 Developers can write web applications in one language, which helps by reducing the context switch between client and server development, and allowing for

code sharing between client and server, such as reusing the same code for form

validation or game logic.

 JSONv is a very popular data interchange format today and is native to JavaScript.

 JavaScript is the language used in various NoSQL databases (such as CouchDBand MongoDB), so interfacing with them is a natural fit (for example, MongoDB’s shell and query language is JavaScript; CouchDB’s map/reduce is JavaScript).

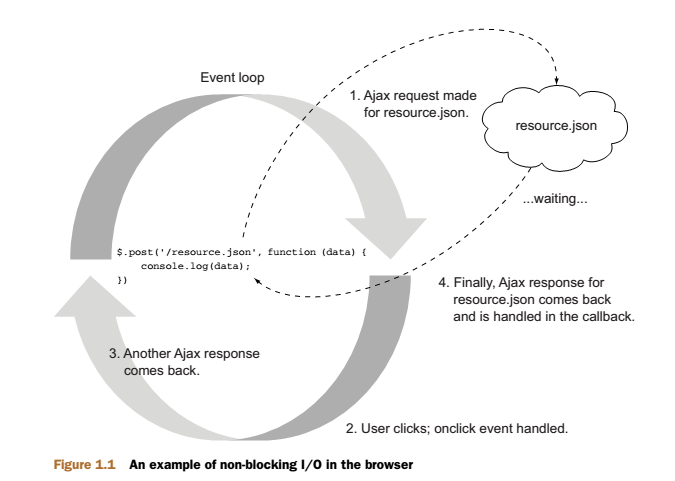
 JavaScript is a compilation target, and there are a number of languages that compile to it already.

 Node uses one virtual machine (V8) that keeps up with the ECMAScript standard.

In other words, you don’t have to wait for all the browsers to catch up to use new JavaScript language features in Node.

Who knew JavaScript would end up being a compelling language for writing server side applications? Yet, due to its sheer reach, performance, and other characteristics mentioned previously, Node has gained a lot of traction. JavaScript is only one piece of the puzzle though; the way Node uses JavaScript is even more compelling. To understand the Node environment, let’s dive into the JavaScript environment you’re most familiar with: the browser.

In this example, the assumption is that the response for resource.json would be stored in the data variable when it is ready and that the console.log function will not execute until then. The I/O operation (the Ajax request) would “block” script execution from continuing until ready. Because the browser is single-threaded, if this request took 400 ms to return, any other events happening on that page would wait until then before execution. You can imagine the poor user experience if an animation was paused or the user was trying to interact with the page somehow. Thankfully, that’s not the case. When I/O happens in the browser, it happens outside of the event loop (outside the main script execution) and then an “event” is emitted when the I/O is finished, which is handled by a function (often called the “callback”) as shown in figure 1.1.

The I/O happens asynchronously and doesn’t “block” the script execution, allowing the event loop to respond to whatever other interactions or requests are being performed on the page. This enables the browser to be responsive to the client and to handle a lot of interactivity on page

**Asynchronous and evented: the server**

For the most part, you’re likely to be familiar with a conventional I/O model for server-side programming, like the “blocking” jQuery example in section 1.2. Here’s an example of how it looks in PHP:

$result= mysql\_query('SELECT\* FROM myTable');

print\_r($result);

This code does some I/O, and the process is blocked from continuing until all the data has come back. For many applications this model is fine and is easy to follow. What may not be apparent is that the process has state, or memory, and is essentially doing nothing until the I/O is completed. That could take anywhere from 10 ms to minutes depending on the latency of the I/O operation. Latency can also result from unexpected causes:

 The disk is performing a maintenance operation, pausing reads/writes.

 A database query is slower because of increased load.

 Pulling a resource from sitexyz.com is sluggish today for some reason.

If a program blocks on I/O, what does the server do when there are more requests to handle? Typically you’d use a multithreaded approach in this context. A common implementation is to use one thread per connection and set up a thread pool for those connections. You can think of threads as computational workspaces in which the processor works on one task. In many cases, a thread is contained inside a process and maintains its own working memory. Each thread handles one or more server connections. Although this may sound like a natural way to delegate server labor—at least

to developers who’ve been doing this a long time—managing threads within an application can be complex. Also, when a large number of threads is needed to handle many concurrent server connections, threading can tax operating system resources.

Threads require CPU to perform context switches, as well as additional RAM. To illustrate this, let’s look at a benchmark (shown in figure 1.2, from http:// mng.bz/eaZT) comparing NGINX and Apache. NGINX(http://nginx.com/), if you aren’t familiar with it, is an HTTP server like Apache, but instead of using the multithreaded approach with blocking I/O, it uses an event loop with asynchronous I/O (like the browser and Node). Because of these design choices, NGINX is often able to handle more requests and connected clients, making it a more responsive solution. In Node, I/O is almost always performed outside of the main event loop, allowing the server to stay efficient and responsive, like NGINX. This makes it much harder for a process to become I/O-bound because I/O latency isn’t going to crash your server or use the resources it would if you were blocking. It allows the server to be lightweight on what are typically the slowest operations a server performs.

Node, unlike many open source platforms, is easy to set up and doesn’t require much in terms of memory and disk space. No complex integrated development environments or build systems are required. Some fundamental knowledge will,

however, help you a lot when starting out. We’ll address two challenges that new Node developers face:

 How to organize your code?

 How asynchronous programming works?

The problem of organizing code is familiar to most experienced programmers. Logic is organized conceptually into classes and functions. Files containing the classes and functions are organized into directories within the source tree. In the end, code is organized into applications and libraries. Node’s module system provides a powerful mechanism for organizing your code, and you’ll learn how to harness it in this chapter. Asynchronous programming will likely take some time to grasp and master; it requires a paradigm shift in terms of thinking about how application logic should execute. With synchronous programming, you can write a line of code knowing that all the lines of code that came before it will have already executed. With asynchronous development, however, application logic can initially seem like a Rube Goldberg machine. It’s worth taking the time, before beginning development of a large project, to learn how you can elegantly control your application’s behavior. In this chapter, you’ll learn a number of important asynchronous programming techniques that will allow you to keep a tight rein on how your application executes.

 How to respond to one-time events

 How to handle repeating events

 How to sequence asynchronous logic

We’ll start, however, with how you can tackle the problem of code organization through the use of modules, which are Node’s way of keeping code organized and packaged for easy reuse.

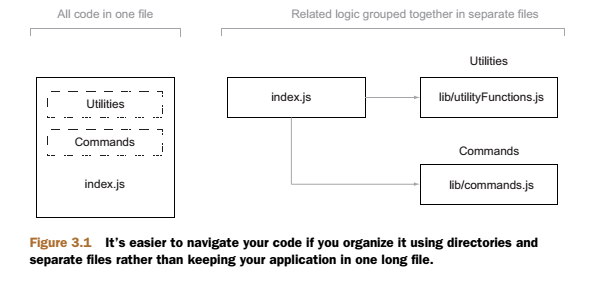
**Organizing and reusing Node functionality**

When creating an application, Node or otherwise, you often reach a point where putting all of your code in a single file becomes unwieldy. When this happens, the conventional approach, as represented visually in figure 3.1, is to take a file containing a lot of

code and try to organize it by grouping related logic and moving it into separate files.

In some language implementations, such as PHPand Ruby, incorporating the logic

from another file (we’ll call this the “included” file) can mean all the logic executed



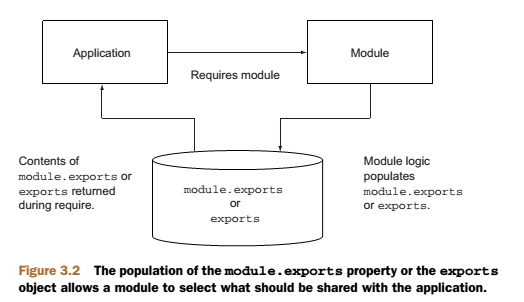
in the included file affects the global scope. This means that any variables created and functions declared in the included file risk overwriting those created and declared by the application. Say you were programming in PHP; your application might contain the following logic:

functionuppercase\_trim($text){ returntrim(strtoupper($text));}include('string\_handlers.php');

If your string\_handlers.php file also attempted to define an uppercase\_trim function you’d receive the following error: Fatal error: Cannot redeclare uppercase\_trim() In PHP you can avoid this by using namespaces, and Ruby offers similar functionality through modules. Node, however, avoids this potential problem by not offering an easy

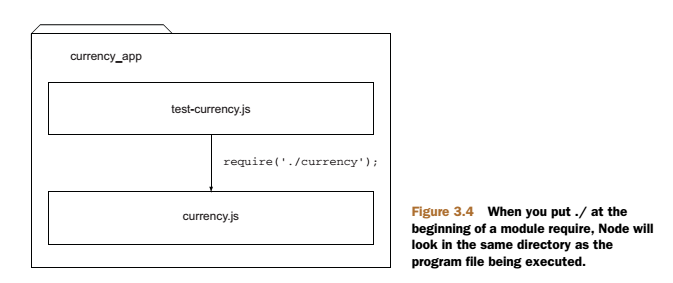
way to accidentally pollute the global namespace. PHP NAMESPACES, RUBY MODULES PHP namespaces are discussed in the manual at http://php.net/manual/en/language.namespaces.php. Ruby modules

are explained in the Ruby documentation: www.ruby-doc.org/core-1.9.3/Module.html. Node modules bundle up code for reuse, but they don’t alter global scope. Suppose, for example, you were developing an open source content management system (CMS) application using PHP, and you wanted to use a third-party API library that doesn’t use namespaces. This library could contain a class with the same name as one in your application, which would break your application unless you changed the class name either in your application or the library. Changing the class name in your application, however, could cause problems for other developers using your CMS as the basis of their own projects. Changing the class name in the library would require you to remember to repeat this hack each time you update the library in your application’s source tree. Naming collisions are a problem best avoided altogether. Node modules allow you to select what functions and variables from the included file are exposed to the application. If the module is returning more than one function or variable, the module can specify these by setting the properties of an object called exports. If the module is returning a single function or variable, the property module .export scan instead be set. By avoiding pollution of the global scope, Node’s module system avoids naming conflicts and simplifies code reuse. Modules can then be published to the npm (Node Package Manager) repository, an online collection of ready-to-use Node modules, and shared with the Node community without those using the modules having to worry



A note about require and synchronous I/O require is one of the few synchronous I/O operations available in Node. Because modules are used often and are typically included at the top of a file, having require be synchronous helps keep code clean, ordered, and readable. But avoid using require in I/O-intensive parts of your application. Any synchronous

call will block Node from doing anything until the call has finished. For example,if you’re running an HTTP server, you would take a performance hit if you used require on each incoming request. This is typically why require and other synchronous operations are used only when the application initially loads.

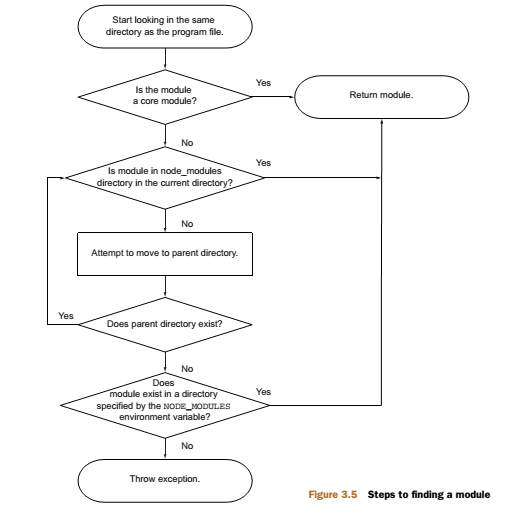


After Node has located and evaluated your module, the require function returns the contents of the exports object defined in the module. You’re then able to use the two functions returned by the module to do currency conversion. If you wanted to put the module into a subdirectory, such as lib, you could do so by simply changing the line containing the require logic to the following:

**var currency = require('./lib/currency');**

Populating the exports object of a module gives you a simple way to group reusable code in separate files. Reusing modules using the node\_modules folder Requiring modules in the file system to exist relative to an application is useful for organizing application-specific code, but isn’t as useful for code you’d like to reuse between applications or share with others. Node includes a unique mechanism for rendering html files.

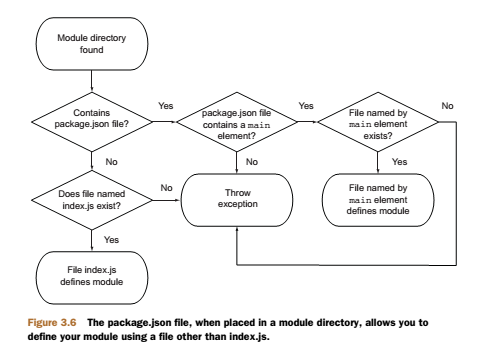
The other thing to be aware of is Node’s ability to cache modules as objects. If two files in an application require the same module, the first require will store the data returned in application memory so the second require won’t need to access and evaluate the module’s source files. The second require will, in fact, have the opportunity to alter the cached data. This “monkey patching” capability allows one module to modify the behavior of another, freeing the developer from having to create a new version of it



# code reuse that allows modules to be required without knowing their location in the file system. This mechanism is the use of node modules directories. In the earlier module example, you required ./currency. If you omit the ./ and simply require currency, Node will follow a number of rules, as specified in figure 3.5, to search for this module.

# The NODE\_PATH environmental variable provides a way to specify alternative locations for Node modules. If used, NODE\_PATH should be set to a list of directories separated by semicolons in Windows or colons in other operating systems.

First, if a module is a directory, the file in the module directory that will be evaluated must be named index.js, unless specified otherwise by a file in the module directory named package.json. To specify an alternative to index.js, the package.json file must contain JavaScript Object Notation (JSON) data defining an object with a key named main that specifies the path, within the module directory, to the main file. Figure 3.6 shows a flowchart summarizing these rules.



# **Main Features**

Following are some of the important features that make Node.js the first choice of software architects.

* **Asynchronous and Event Driven** All APIs of Node.js library are asynchronous that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call.
* **Very Fast** Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
* **Single Threaded but Highly Scalable** - Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way and makes the server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
* **No Buffering** - Node.js applications never buffer any data. These applications simply output the data in chunks.

# **Interface**

Node.JS interface is classified into modules,

Node.JS- a Common.JS Module Implementation.

Your code uses require to include modules.

Modules use exports to make things available.

**COMMON.JS**

An ecosystem for JavaScript outside the browser

1. Modules
2. Promises
3. Binary
4. Filesystem
5. Console
6. System
7. Testing

# **Application**

The module exports two specific components:

A Console class with methods such as console.log (), console.error () and console.warn () that can be used to write to any Node.js stream.

A global console instance configured to write to stdout and stderr. Because this object is global, it can be used without calling require('console').

**Example using the global Console:**

console.log('hello world');

// Prints: hello world, to stdout

console.log('hello %s', 'world');

// Prints: hello world, to stdout

console.error(new Error('Whoops, something bad happened'));

// Prints: [Error: Whoops, something bad happened], to stderr

const name = 'Will Robinson';

console.warn(`Danger ${name}! Danger!`);

// Prints: Danger Will Robinson! Danger!, to stderr

**Example using the Console class:**

const out = getStreamSomehow();

const err = getStreamSomehow();

constmyConsole = newconsole.Console(out, err);

myConsole.log('hello world');

// Prints: hello world, to out

myConsole.log('hello %s', 'world');

// Prints: hello world, to out

myConsole.error(new Error('Whoops, something bad happened'));

// Prints: [Error: Whoops, something bad happened], to err

const name = 'Will Robinson';

myConsole.warn(`Danger ${name}! Danger!`);

// Prints: Danger Will Robinson! Danger!, to err

The API for the Console class is designed fundamentally around the Web browser Console object, the Console is Node.js is not intended to duplicate the browsers functionality exactly.

# **7.3 What is Ajax**

Ajax is not a programming language or a tool, but a concept. Ajax is a [client-side script](http://www.seguetech.com/blog/2013/02/07/what-are-the-pros-and-cons-of-client-side-scripting) that communicates to and from a server/database without the need for a[postback](http://www.c-sharpcorner.com/uploadfile/2f73dd/what-is-postback-in-Asp-Net/) or a complete page refresh. The best definition for Ajax is “the method of exchanging data with a server, and updating parts of a web page - without reloading the entire page.” Ajax itself is mostly a generic term for various JavaScript techniques used to connect to a web server dynamically without necessarily loading multiple pages. In a more narrowly-defined sense, it refers to the use of [XmlHttpRequest](http://en.wikipedia.org/wiki/XMLHttpRequest) objects to interact with a web server dynamically via JavaScript.

AJAX describes the programming paradigm that combines JavaScript and a web server . Developers use AJAX to create highly interactive web applications such as Microsoft Virtual Earth . Without AJAX, a web application might make the visitor wait while a response is gathered from the web server . An AJAX-based application sends requests from the web browser to the web server in the background (asynchronously) while the visitor is using the application . This makes the application feel much more responsive to the user . In an AJAX application, JavaScript processes the response and presents the data to users . When combined with Cascading Style Sheets (CSS) and a good layout, an AJAX application provides excellent usability and the portability that only a web application can . As complex as some AJAX applications may seem, the actual process of sending a request and handling the response are not terribly complicated . This chapter explores how you can send requests and receive responses using a fundamental AJAX object: XMLHttpRequest . One central concept in AJAX is that you call server-side applications to return data . In this chapter, I give you a brief overview of how to create such an application using both ASP .NET and PHP . (PHP is a recursive acronym for PHP Hypertext Preprocessor .) If you need additional assistance in creating the server-side portion of an AJAX application, you can get help from several sources.

**The XMLHttpRequest Object**

The XMLHttpRequest object is central to building an AJAX application . Although implementations of JavaScript differ, the ECMAScript and the World Wide Web Consortium (W3C) have standardized many aspects of it except the XMLHttpRequest object, which has never been subject to a standardization process . Even so, since the release of Windows Internet Explorer 7, you use the XMLHttpRequest object in the same way across all major browsers . Microsoft first implemented the XMLHttpRequest object in Microsoft Internet Explorer 5 .0 . If a visitor is using a browser version earlier than that, applications using XMLHttpRequest won’t work . In Internet Explorer versions prior to version 7, the XMLHttpRequest object was instantiated as an ActiveXObject object, but other browsers implemented the XMLHttpRequest object as a JavaScript object built into the browser . This means that if your applications need to work with versions of Internet Explorer earlier than version 7, you need to instantiate the XMLHttpRequest object for those browsers in a different way, as I show you later in the chapter . The next section, “Instantiating the XMLHttpRequest Object,” shows how you can test for the existence of XMLHttpRequest and how to instantiate it in all versions of Internet Explorer .

Instantiating the XMLHttpRequest Object Internet Explorer 7 and later versions, and all other major browsers that support XMLHttpRequest, instantiate the XMLHttpRequest object in the same way:

**var req = new XMLHttpRequest();**

**Sending an AJAX Request**

With a newly created XMLHttpRequest object in hand, you can send requests to the web server and get responses . To send the request, you use a combination of the open() and send() methods of the XMLHttpRequest object . There are two fundamentally different ways to send AJAX requests: synchronously and asynchronously . When sent in a synchronous manner, the requesting code simply waits for the response—a process called blocking . So, for a synchronous request, the requesting code will block, effectively preventing further processing or execution of other JavaScript while the script waits for the response from the web server . This process has obvious disadvantages when the request or response gets lost in transit or is just slow . With asynchronous requests, the requesting code doesn’t block . Instead, the caller can check the request status to discover when the request has completed . You see more about asynchronous requests later in this chapter; it’s easier to work with synchronous requests first . Before you can send a request, you have to build it . To do that, you use the open method, which has three arguments: the request method (GET, POST, HEAD, or others), the Uniform Resource Locator (URL) to which the request will be sent, and a Boolean true or false, indicating whether you want to send the request asynchronously or synchronously, respectively.

Processing an AJAX Response It’s easier to work with the response when the request is sent synchronously, because the script’s execution stops while awaiting the response . The requestObj variable provides helpful methods for processing a response, including giving access to the status codes and text of the status sent from the server . Regardless of whether the request is synchronous or asynchronous, you should evaluate the status code to ensure that the response was successful (usually indicated by a status of 200) . The responseText method contains the text of the response as received from the web server . For example, assume a server application returns the sum of two numbers . Calling the application to add the numbers 2 and 56 looks like this:

Here’s a synchronous call and response retrieval:

requestObj.open("GET","http://www.braingia.org/addtwo.php?num1=2&num2=56", false); requestObj.send();

if(requestObj.status==200)

{ alert(requestObj.responseText); }

else { alert(requestObj.statusText); }

In this example, assume that the requestObj was created using the readyAJAX() function that you saw earlier . The preceding code then calls the open method using a GET request to the specified URL (http://www.braingia.org/addtwo.php?num1 2&num2 56) . The request is sent synchronously because the last argument to the open method is false . Next, the code calls the send method, which actually sends the request to the web server . When the client receives the response from the web server, it calls the status method to check the status value. If the response code is 200, indicating success, the code displays the response Text, which holds the response from the server. If the response status code is anything other than 200, the code displays the status text. Processing an asynchronous response is a bit more complex. When a request is sent asynchronously, script execution continues. Therefore, it is unpredictable when the script will be notified that the response has been received. To know the response status, you can use the on ready state change event to trigger code that checks the event’s ready State property to determine the state of the request/response cycle.

**Working with JSON**

JavaScript Object Notation (JSON) is a way to pass data as native JavaScript objects and arrays, rather than encode data within XML (or HTML) responses. JSON is a more efficient way to pass data from server to client. Parsing XML using the DOM is more complex and thus slower, whereas parsing JSON-encoded data is done directly in JavaScript. Recall the book .xml document from an earlier example in this chapter. That same data in JSON looks like this:

{"book": { "title": "JavaScript Step by Step", "isbn": "9780735624498" }}

Retrieving an individual element is somewhat easier with JSON than with XML. You use the JavaScript eval() function to parse the JSON-formatted response . For example, here’s the code to retrieve and display the book title:

var requestObj = readyAJAX();

var url = "http://www.braingia.org/json.php";

requestObj.open("GET",url,false);

requestObj.send();

if (requestObj.status == 200)

{var xmldocument = eval(‘(‘ + requestObj.responseText + ‘)'); alert(xmldocument.book.title); } else { alert(requestObj.statusText); }

Using JSON carries an inherent security risk, because it uses the eval() function to parse the response . The eval() function essentially executes the JavaScript code received, so if that code were malicious, it would execute in the context of the application being run . It is your responsibility to ensure that the data your application is using with JSON is clean and free of malicious code that could cause problems when executed using eval() .

**Processing Headers**

The HTTP HEAD method returns just the response headers from the server, rather than the headers and the body in the way the GET method does . The HEAD method is sometimes helpful for determining whether a given resource has been updated or changed . One frequently-sent HTTP header is Expires, which indicates when the client should request a refreshed copy of a document rather than read it from the client’s cache . If the server sends the Expires header, the HEAD method is an efficient way to view and parse the Expires header because the HEAD method retrieves only the response header rather than the entire body of the requested resource . To request only the response headers from a server, whether using a HEAD request or any other type of request such as GET or POST, use the getAllResponseHeaders() method of the XMLHttpRequest object, as follows:

requestObj.getAllResponseHeaders();

# **Benefits of Ajax**

There are 4 main benefits of using Ajax in web applications:

**Callbacks:** Ajax is used to perform a callback, making a quick round trip to and from the server to retrieve and/or save data without posting the entire page back to the server. By not performing a full postback and sending all form data to the server, network utilization is minimized and quicker operations occur. In sites and locations with restricted bandwidth, this can greatly improve network performance. Most of the time, the data being sent to and from the server is minimal. By using callbacks, the server is not required to process all form elements. By sending only the necessary data, there is limited processing on the server. There is no need to process all form elements, process the ViewState, send images back to the client, or send a full page back to the client.

**Making Asynchronous Calls:** Ajax allows you to make asynchronous calls to a web server. This allows the client browser to avoid waiting for all data to arrive before allowing the user to act once more.

**User-Friendly:** Because a page postback is being eliminated, Ajax enabled applications will always be more responsive, faster and more user-friendly.

**Increased Speed:** The main purpose of Ajax is to improve the speed, performance and usability of a web application. A great example of Ajax is the movie rating feature on Netflix. The user rates a movie and their personal rating for that movie will be saved to their database without waiting for the page to refresh or reload. These movie ratings are being saved to their database without posting the entire page back to the server.

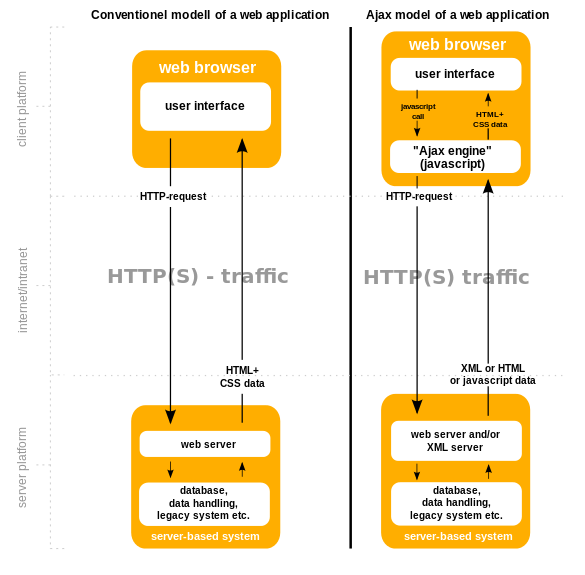
# **Technical Aspects**

Ajax callbacks can be done by instantiating an XMLHttpRequest object in the client-side JavaScript. The XMLHttpRequest object can be used to directly call server-side objects like pages and web services. These pages and web services will either save and/or return data.

Ajax was originally an acronym for Asynchronous JavaScript and XML. “Asynchronous” means that multiple events are happening independently of one another. Once a client initializes an Ajax callback to the server, the client will not need to wait for a response and can continue to use the web application while the request is being processed. Once done, the server will send a response back to the client and the client will process it as necessary.

# **Application**

Applications created with Ajax use an [engine](http://whatis.techtarget.com/definition/engine)that acts as an intermediary between a user's browser and the server from which it is requesting information. Instead of loading a traditional Web page, the user's browser loads the Ajax engine, which displays the page the user sees. The engine continues to run in the background, using JavaScript to communicate with the Web browser. User input or clicking on the page sends a JavaScript call to the Ajax engine, which can respond instantly in many cases. If the engine needs additional data, it requests it from the server, usually using XML, while it is simultaneously updating the page.



**Example**

Here is an example of a simple Ajax request using the [GET](https://en.wikipedia.org/wiki/GET_(HTTP)) method, written in [JavaScript](https://en.wikipedia.org/wiki/JavaScript).

get-ajax-data.js:

*// This is the client-side script.*

*// Initialize the Http request.*

**var**xhr=**new**XMLHttpRequest ();

xhr.open ('get','send-ajax-data.php');

*// Track the state changes of the request.*

xhr.onreadystatechange=**function**(){

**var**DONE=4;*// readyState 4 means the request is done.*

**var**OK=200;*// status 200 is a successful return.*

**if**(xhr.readyState===DONE){

**if**(xhr.status===OK){

alert (xhr.responseText);*// 'This is the returned text.'*

}**else**{

alert ('Error: '+xhr.status);*// An error occurred during the request.*

}

}

};

*// Send the request to send-ajax-data.* xhr.send (**null**);

# **7.4 What is Mongo DB?**

MongoDB is one of the [database](http://searchsqlserver.techtarget.com/definition/database) type which lies under the [NoSQL](http://searchdatamanagement.techtarget.com/definition/NoSQL-Not-Only-SQL) banner. Instead of using [tables](http://searchsoa.techtarget.com/definition/table) and [rows](http://searchoracle.techtarget.com/definition/row) as in [relational databases](http://searchsqlserver.techtarget.com/definition/relational-database), MongoDB is built on anarchitecture of collections and documents. Documents comprise sets of [key-value pairs](http://searchenterprisedesktop.techtarget.com/definition/key-value-pair) and arethe basic unit of data in MongoDB. Collections contain sets of documents and function as the equivalent of relational database tables.

MongoDB supports dynamic [schema](http://searchsqlserver.techtarget.com/definition/schema) design, allowing the documents in a collection to have different fields and structures. The database uses a document storage and data interchange format called BSON, which provides a binary representation of [JSON](http://searchwindevelopment.techtarget.com/definition/JSON-Javascript-Object-Notation)-like documents. Automatic [sharding](http://searchcloudcomputing.techtarget.com/definition/sharding) enables data in a collection to be distributed across multiple systems for horizontal as data [volumes](http://searchstorage.techtarget.com/definition/volume) increase.

We are going to take a look at the technology stack we'll be using throughout the book. After a brief discussion of Node, npm, MongoDB, and Express we will introduce Mongoose as an **ODM**(**Object-Document Modeler**), cover good and bad use cases, and introduce the two cornerstones of Mongoose.

By the end of this chapter you will have an understanding of the technology stack and where Mongoose fits in. We will also have set up the sample project that we will build throughout the book.

**The technology stack – Node.js, npm, MongoDB, and Express**

The benefits of Node are making it ever more popular, and a set of compatible technologies are fast becoming the basis of a new standard development stack.

**The language and the server – JavaScript and Node**

For quite some time JavaScript was mainly thought of as a lightweight browser- based scripting language. Microsoft did support a JavaScript version of Classic ASP, but it was largely side-lined in favor of the VBScript version.

Fast-forward 10 years and there were some pretty impressive JavaScript-basedapps on the Web, for example Gmail. The general view of JavaScript as a programming language was starting to change back then.

In 2010 a new server-side system called **Node.js**was starting to make waves in the Internet developer community. Node.js was a new way for using JavaScript on the server side again. And it was quick and efficient. It could make scaling a web

application much more cost-effective by reducing the amount of hardware requiredper-site visitor or request.

By 2012 Node was one of the buzzwords of the start-up scene, and you can see why.

Firstly, most Web developers have some JavaScript experience so it doesn't require you to learn a new language from scratch. This also means you can share some code betweenfront-end and back-end, so you don't have to code the same thing twice in two different languages. An excellent example of this is form validation; you want real-time validation in the browser for a better user experience, but you also need to validate on the server side to protect your system. So you code the same thing twice. Using Node you can use the same validation script on the server side and the browser, so you only have to code it once, in JavaScript.

Second, there is the reduced cost of scaling, especially when dealing with large numbers of concurrent users. On the same hardware Node's efficiencies allow it to handle many more requests than classic stacks on Apache or IIS. That being said, adding scalability to Node is more complicated than other stacks. Unlike the others you can't just put it on a more powerful machine and set it running. By default

Node will currently only run one process, using only one core of a machine. There are methods to address this issue, using a load balancer in front of several processes running alongside each other for example, and there are plans to enable future versions of Node to be able to manage this natively, directly addressing this issue.

The benefits of scalability do have a cost. The single process is a more complicated approach to server-side programming and requires a change in mindset.

**Single-threaded versus multithreaded**

Traditional stacks are generally **multithreaded**. This means that every new visitor or session is given a new thread, and these are never shared. One session's activity generally doesn't impact another, until the server resources are exhausted. For example, if Session 1 is doing a complex database write operation it may take

a couple of seconds, but Session 2 continues running oblivious to this.

Node is **single-threaded**. This means that every visitor or session is added to that one thread. So it is possible for a two-second database write operation to hold up every other user for two seconds. Multiply this by just 10 users and you've got a big problem on your hands.

Addressing this requires a different way of coding.

**Blocking versus non-blocking code**

In the traditional stack, the approach to coding would be one step after the other, as in the following steps:

1. First, take the data.

2. Then write this data to the database.

3. Send a confirmation message.

4. Wait for the next request.

This is **blocking**code, as you can only do one thing at a time. This is fine in the multithreaded stack, as you're only ever responding to one person's requests.

In the single-threaded stack, you may have to respond to several people's requests at the same time, so you can't afford to be stuck doing time-consuming operations or waiting for someone else to do something. To do this, the approach to coding becomes more like the following:

1. You give this data to someone.

2. They write this data to the database.

3. When they are done, they send a confirmation message; if this isn't something they can do, then they add it to your request list.

4. You're going to take the next request.

This is **non-blocking**code. You can only do one at a time. So you're getting someone else to do something for you, and telling them what to do when they have finished.

You can then deal with the next request coming in without any delay.

**JavaScript callbacks**

The way to code this in JavaScript is to use callbacks. Most JavaScript coders start using them before they even know it, particularly anybody who uses libraries such as jQuery.

Take the basic jQuery document.ready method as shown in the following:

$(document).ready(function() { console.log("document ready");

});

This is an event driven callback. The $(document).ready() part is a method function of jQuery, and we are sending it a function function() that it can run at the appropriate time. So we are saying "Hi ready, here is what I want you to do once the document is ready, I'll leave it up to you to decide when that is". The callback function we are using in this example is the following code snippet:

function() { console.log("document ready");

}

**Running the callback**

The jQuery .ready() function is pretty complicated, so we're not going to look at that here. However, the construct is very useful to understand. Look at the following code snippet:

ready = function (callback) {

//do something

//do something else

//....

//and so on callback();

};

So ready itself is a function, and this function accepts one parameter callback. The callback parameter is generally an anonymous function, like the one we looked at earlier. A very important point to note is that callback now exists in the scope of the ready function. This means that your callback function has access to any variables or objects created in the ready function.

**A Node.js example**

Now consider the following standard Node "hello world" example:

var http = require('http'); http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/plain'

});

res.write('Hello world'); res.end();

})

listen(8888, '127.0.0.1');

Look familiar? This is sending a callback to the http.createServer method function. See how the parameters—req and res—are being sent to the callback function even though they haven't been defined or created anywhere. This works because the http.createServer function will create these objects before calling this callback function that will use them.

**The database – MongoDB**

MongoDB has become the main database of choice for working with Node. Note that there are Node drivers for many other databases, including MySQL, Microsoft SQL Server, Reddis, PostgreSQL, CouchDB, and more.

MongoDB is popular as it is fast and flexible with excellent community support. It is a document-oriented database, fitting in somewhere between Key-Value stores and traditional relational databases. Despite being a document store, MongoDB also enables rich querying and secondary indexing of documents, setting it apart from other databases and making it a very powerful option.

MongoDB stores documents as BSON, which is effectively binary-encoded JSON.

When you run a query you get a JSON object returned (or a string in JSON format, depending on the driver). Look at the following code snippet for example:

{"\_id" : ObjectId("4ffbc45c35097b5a1583ad71"), "firstname" : "Simon", "lastname" : "Holmes }

So, a document is a set of keys (for example, firstname) and values (for example,

Simon). The \_id entry is a unique identifier that the underlying MongoDB driver will—by default—create for each new document.

If you are more experienced with relational databases, it may help you to think of a document as a bit like a row in a table. In this analogy, the **key**can be thought of as a column. An important difference is that each document doesn't have to contain the exact same set of keys, and there is no direct need to have keys with empty values taking up space.

A collection of documents is called a **collection**. The closest analogy is a table. So in your database you could well have multiple collections, such as a users collection, posts collection, and stats collection.

MongoDB is also extremely scalable, with many built-in capabilities for distributing across multiple servers, without compromising speed or data integrity.

With everything combined, it makes MongoDB a great playmate for Node.

**The framework – Express**

**Express**is a web application framework for Node.

When you create a Node project, you have to do a lot more groundwork than you might be used to. Until you create it, there isn't even a web server. Then you have to deal with serving static files in a non-blocking way, figure out the routing mechanism, view engine, session management, and so on.

Or you can create an Express project and let it do all of this for you, in a tried-and- tested way. At the end of this chapter, we'll see how easy it is to set up an

Express project.

Note that Express is not required to build a Node application, but it is a great starting point for building web applications.

**What Mongoose is all about?**

Mongoose is an object modeling tool for MongoDB and Node.js. What this means in practical terms is that you can define your data model in just one place, in your code.

Yes, that's right. You don't have to create a schema in the database, link that to an ORM or map it into your project objects and classes. You can just define your data structure in JSON inside your project.

The first time I created a project like this I was amazed at how much time and frustration it saves. Even now I still get that warm glow when I start a new project or prototype using Mongoose. It's like taking a shortcut to work down deserted country lanes while everybody else is gridlocked on the highway.

A schema definition can be as simple as the following code snippet:

var userSchema = new mongoose.Schema({ firstname: String,

lastname: String, createdOn: Date

});

A document in MongoDB created from this schema would be like the following code snippet:

{"\_\_v" : 0, "\_id" : ObjectId("51412597e8e6d3e35c000001"), "createdOn" : ISODate("2013-03-14T01:19:19.866Z"),"firstname" : "Simon", " lastname " : "Holmes" }

If you want to refactor, then you can just do it from within your code, saving a huge amount of development time.

**What is Mongoose good for?**

Mongoose is primarily useful when you want to interact with structured data in

MongoDB. It allows you to define a schema for your data, so that you can interact with your MongoDB data in a structured and repeatable way.

Mongoose helps with many common MongoDB tasks, and removes some of levels of complexity from the nested callbacks you find yourself getting lost in with the native

MongoDB driver.

Mongoose also returns the data to you as a JSON object that you can use directly, rather than the JSON string returned by MongoDB.

Mongoose also has a whole suite of helper functions and methods that we'll explore throughout the subsequent chapters of this book.

**What Mongoose is not ideally suited for**

Mongoose is probably not the answer for you if you are primarily working with the following:

•Schema-less data

•Random documents

•Pure Key-Value pairs

**The cornerstones of Mongoose**

There are two aspects of Mongoose that we need to introduce you to before going much further:

•Schemas

•Models

This is a very high-level view; so don't worry if you're not 100 percent confident with this just yet, as we'll be covering it in a lot more detail later in this book.

**Mongoose schemas**

As we saw earlier, a **schema**is fundamentally describing the data construct of a document. This schema defines the name of each item of data, and the type of data, whether it is a string, number, date, Boolean, and so on.

var userSchema = new mongoose.Schema({ name: String,

email: String, createdOn: Date, verified: Boolean

});

In most scenarios you would have one schema for each collection within the database.

Schemas are a powerful aspect of Mongoose, which can also be extended with helper functions and additional methods. But we'll describe more about that in a later chapter.

**Mongoose models**

A **model**is a compiled version of the schema. One instance of the model will map to one document in the database.

Creating a User instance based on the schema userSchema is a one line task:

var User = mongoose.model('User', userSchema);

It is the model that handles the reading, creating, updating, and deleting of documents.

**Installing MongoDB**

Again we're using APT here.

**$ sudo apt-get install mongodb**

In order to test MongoDB, we need to run it. You can do this by just typing mongodb into terminal, but I prefer to run it as a persistent service so that you don't have to restart it.

**$ sudo service mongodb start**

This should give you a confirmation message in terminal that the mongodb process is running.

Now to finally test that it has installed correctly we'll drop into the built-in MongoDB shell. In terminal, enter the following:

**$ mongo**

This should do three things:

•Enter you into the MongoDB shell

•Show you the version of the shell being used

•Connect to the test database that comes with MongoDB

We are going to look at the two methods Mongoose uses to connect to databases, mongoose.connect and create Connection. We will also go through various configuration options.

**Mongoose default connection**

The way to set the default connection to a MongoDB database via Mongoose is nice and easy, using the mongoose.connect method.

var dbURI = 'mongodb://localhost/mydatabase'; mongoose.connect(dbURI);

This will open a Mongoose connection to the Mongo database mydatabase, running on the server localhost. If established at the correct place in your code, this connection will now be available at any point in your app, if you *require*Mongoose.

**Using multiple connections**

The default connection is great if your app only needs to connect to one database. But what happens if you need to connect to a second database at the same time? Or connect to the same database as a different user with different permissions.

For this we can use the mongoose.create Connection method, which is as follows in the code snippet:

var dbURI = 'mongodb://localhost/myadmindatabase';

var adminConnection = mongoose.createConnection(dbURI);

**About the connection string**

The connection string—dbURI in our examples—can use any of the options from the MongoDB connection string. We're not going to cover them all here, but here are a couple of common options.

**Setting the port**

The default port is 27017. If you wanted to specify a different port, say 27018, you would do so using the following code snippet:

var dbURI = 'mongodb://localhost:27018/mydatabase';

**Specifying a database user**

If you want to access the database as a particular user, you can add the username and password in front of the hostname.

var dbURI = 'mongodb://username:password@localhost/mydatabase';

**Connection options**

Mongoose allows you to pass certain options through to both connection methods.

If specified, these options will override those set in the connection string.

The options are sent as a JSON object as an optional second parameter to the connection call. For example:

var dbURI = 'mongodb://localhost/mydatabase';

var dbOptions = {'user':'db\_username','pass':'db\_password'}; mongoose.connect(dbURI, dbOptions);

The options you can use this way are:

•**user and pass**: Username and password for the database, if required and not specified in the connection string.

•**db**: This relates to the DB options available in the Node MongoDB Native driver.

•**server**: This relates to the server options available in the Node MongoDB Native driver.

•**replset**: This option allows you to specify a ReplicaSet. The details of ReplicaSet methods are beyond the scope of this book, but the principle is that you can have one primary database where all the writes are made to, and multiple secondary databases. If the primary database fails, one of the secondary databases is automatically made the new primary.

**Closing the connection**

As we have already seen, the general best practice is to open your connection at application start up, and keep it open. However, there are times when you will want to close the connection. For example, if your application is shutting down, or restarting, the database connection needs to be manually closed, or if you are running a single-hit script rather than a persistent application.

**Calling the close command**

Each Mongoose connection has a close() method that takes an optional callback function.

If you are using the default connection you call it like the following code snippet does:

mongoose.connection.close(function () { console.log('Mongoose default connection closed');

});

Calling the close() method on a named connection is just as easy, using our example from earlier:

adminConnection.close(function () {

console.log('Mongoose connection adminConnection closed'); });

*Establishing a Database Connection*

**Closing when the Node process ends**

As a rule you should tidy up the connections when your Node application stops, whether this is an intentional process termination or not. To do this, you can send your disconnections as a callback to Node's process.on('SIGINT') event as shown in the following:

process.on('SIGINT', function() { mongoose.connection.close(function () {

console.log('Mongoose disconnected through app termination'); process.exit(0);

});

});

**Connection events**

The connection process in Mongoose inherits the Node EventEmitter class, meaning that we can set certain code to run following specific events. We can—andwill—work with some of the connection events, such as connected,disconnected, and error.

The connection events are all used in the same way, sending a callback to theconnection.on event listener. For example, if we wanted to log to the console when a connection error occurs we could do this in the following code:

mongoose.connection.on('error',function (err) { console.log('Mongoose connection error: ' + err);

});

**Connecting our project**

Now we know what we're doing, let's connect our project to a database using the default Mongoose connection.

**Creating the connection**

For the sake of well organized code, let's create a folder called model, and in that, an empty JavaScript file called db.js. We'll use this for managing the Mongoose connection, and will add to it in later chapters.

At this stage the file needs to do three things:

1. Bring in the Mongoose module

2. Build the connection string for the database

3. Open the Mongoose connection to the database

So in your /model/db.js file, enter the following:

//Bring Mongoose into the project var mongoose = require( 'mongoose' );

//Build the connection string

var dbURI = 'mongodb://localhost/MongoosePM';

// Create the database connection mongoose.connect(dbURI);

Each of the three objectives is achieved with just one line of code—pretty simple don't you think!

**Catching the events**

Next up we want to set up our event handlers. At this stage, we are just going to log messages to the console, but they are useful containers, and important to understand. We will also catch when the Node process is ending and close the

Mongoose connection.

So, still in db.js, after the connection code add the following snippets:

mongoose.connection.on('connected', function () { console.log('Mongoose connected to ' + dbURI);

});

mongoose.connection.on('error',function (err) { console.log('Mongoose connection error: ' + err);

});

mongoose.connection.on('disconnected', function () { console.log('Mongoose disconnected');

});

process.on('SIGINT', function() { mongoose.connection.close(function () {

console.log('Mongoose disconnected through app termination'); process.exit(0);

});

});

*Establishing a Database Connection*

**Opening the connection at application start**

Now we need to tell the project to use this file, so that we can connect to the database when the application starts. This is as simple as requiring our new file inapp.js— you may remember that this is the file we run to start the application.

At the top of your app.js file you should see some of the default modules being required like in the following:

var express = require('express') , routes = require('./routes')

We are simply going to add one line in here, to require our model/db.jsfile immediately after express is required. You should end up with something resembling this following code:

var express = require('express')

,db = require('./model/db')

,routes = require('./routes')

**Creating the database**

You will notice that we haven't actually created a MongoDB database yet. The good and perhaps surprising—news is that we don't have to. Nor do we have to explicitly create any collections.

MongoDB will create a database the first time anything is saved to it, and the same for collections.

var dbURI = 'mongodb://localhost/MongoosePM';

// Create the database connection mongoose.connect(dbURI);

mongoose.connection.on('connected', function () { console.log('Mongoose connected to ' + dbURI);

});

# **Main Features**

Some of the features include:

* **Document-oriented**

Instead of taking a business subject and breaking it up into multiple relational structures, MongoDB can store the business subject in the minimal number of documents. For example, instead of storing title and author information in two distinct relational structures, title, author, and other title-related information can all be stored in a single document called Book.

* **Ad hoc queries**

MongoDB supports field, range queries, regular expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions.

* **Indexing**

Any field in a MongoDB document can be [indexed](https://en.wikipedia.org/wiki/Database_index) – including within arrays and embedded documents (indices in MongoDB are conceptually similar to those in[RDBMSes](https://en.wikipedia.org/wiki/RDBMS)). Primary and secondary indices are available.

* **Replication**

MongoDB provides high availability with replica sets. A replica set consists of two or more copies of the data. Each replica set member may act in the role of primary or secondary replica at any time. The primary replica performs all writes and reads by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can optionally perform read operations, but that data is eventually consistent by default.

* **Load balancing**

MongoDB scales horizontally using [sharding](https://en.wikipedia.org/wiki/Sharding). The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more slaves.). Alternatively, the shard key can be hashed to map to a shard – enabling an even data distribution.

MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. MongoDB is easy to deploy, and new machines can be added to a running database.

* **File storage**

MongoDB can be used as a [file system](https://en.wikipedia.org/wiki/File_system), taking advantage of load balancing and data replication features over multiple machines for storing files.

This function, called [Grid File System](https://en.wikipedia.org/wiki/Grid_File_System), is included with MongoDB drivers and available for many development languages (see "[Language Support](https://en.wikipedia.org/wiki/MongoDB#Language_support)" for a list of supported languages).

In a multi-machine MongoDB system, files can be distributed and copied multiple times between machines transparently, thus effectively creating a load-balanced and fault-tolerant system.

* **Aggregation**

[MapReduce](https://en.wikipedia.org/wiki/MapReduce) can be used for batch processing of data and aggregation operations.

The aggregation framework enables users to obtain the kind of results for which the [SQL](https://en.wikipedia.org/wiki/SQL) GROUP BY clause is used. Aggregation operators can be strung together to form a pipeline – analogous to [Unix pipes](https://en.wikipedia.org/wiki/Pipeline_(Unix)). The aggregation framework includes the $lookup operator which can join documents from multiple documents.

* **Server-side JavaScript execution**

JavaScript can be used in queries, aggregation functions (such as MapReduce), and sent directly to the database to be executed.

* **Capped collections**

MongoDB supports fixed-size collections called capped collections. This type of collection maintains insertion order and, once the specified size has been reached, behaves like a [circular queue](https://en.wikipedia.org/wiki/Circular_queue).

**Conclusion**

The main reason for doing this project is solve the parking problem of our university. In University of Karachi there are limited gates and number of vehicles passing through each gate, which when checked manually raises inconvenience and also to depot five to six persons on each gate. This problem is aimed to solve through this computer program.

**References**

* <http://www.owlnet.rice.edu/~elec539/Projects97/morphjrks/moredge.html>
* <https://sisu.ut.ee/imageprocessing/book/1>
* <http://www.mathworks.com/discovery/edge-detection.html?refresh=true>
* <http://www.engineersgarage.com/articles/image-processing-tutorial-applications>
* <https://nodejs.org/en/docs/>
* <https://docs.mongodb.org/manual/contents/>
* <https://docs.mongodb.org/manual/core/crud-introduction/>
* <https://docs.mongodb.org/manual/reference/database-references/>
* <https://github.com/gulpjs/gulp/blob/master/docs/getting-started.md>
* <https://developer.mozilla.org/en-US/docs/AJAX/Getting_Started>
* <http://bootboxjs.com/#getting-started>
* <http://getbootstrap.com/components/>