# Chapter 1.

# Systems for pilot traning and pilot evaluation.

## Flight simulators.

### Basic description of flight simulators.

### Types of flight simulators.

All available aircraft simulators can be divided into two main types:

* Software simulators;
* Training complexes.

In modern terminology, aviation simulators with a fixed cabin belongs to flight simulators. Simulators significantly differ in design depending on the destination: from mechanics and electronic equipment from the dashboard and the front part of the fuselage, designed to train pilots to computer programs PCs. Many software simulation realism is characterized as low because it does not allow the use of all the senses and is used in gaming purposes for personal computers.

Software simulators divided into procedural and comprehensive. *Procedural* aircraft simulator designed for training flight crews. This technical teaching tool that allows you to shape the skills needed in the real world. It has the following main features: simulator imitation of individual fragments of conditions of real activity pilot; the possibility of practicing in certain operations and actions of real pilot with cab equipment; the possibility of objective monitoring results of all operations, practiced on the simulator and instructor’s actions.

The procedural simulators provide training of specific actions, such as control of the aircraft, engines and aviation systems, staff, management of electronic equipment, combat use and so on. Typically, this kind of simulators are composed of display boards and instrument simulators and simulators control levers, whose boundary movements, load characteristics and tactile sensations correspond to real at all stages and modes of flight. Some devices that are closest to the operation are real.

The procedural simulators designed for working crew procedures and training for the flight. Purpose consoles, instruments and controls are generally simulated using touch monitors. For the convenience of individual panels and controls can be presented as full-size models. Additionally, depending on the amount of realized tasks, training can be divided into the following types:

1. Functional (primary) cabins, which are modeled to display information controls. They make it possible to deepen the knowledge of students-pilots of aerodynamics and aviation equipment, off procedure during the pilot operation of aircraft. Primary aviation simulators are usually the simplest, often made by the aviation units and schools. Stands and models can be considered as functional simulators.
2. Specialized training designed to prepare cadets-pilots for doing specific activities, for development of certain psychological qualities and skills of action in special cases in flight.

A *comprehensive* aviation training simulator implements similar to procedural simulator, but on advanced level and has such basic features as approaching the maximum conditions of the pilot in the simulator to the real conditions of the flight. Providing practice on the simulator in general of all tasks of a real pilot, which he carries in flight; enable objective monitoring results of all tested tasks.

An integrated simulator - the highest level of technical training to prepare flight crews and effective means of maintaining trained skills of pilots. An integrated simulator recreates real cab interior also makes it possible to work out all modes of operation of the aircraft. Simulators of highest qualification level have complete set of tools that provide adequate performance in all channels of perception cadet.

### The Concept of Real-time Simulation.

Typically, office operating systems provide acceptable control of the mouse and a mechanism for ensuring that the software is activating mouse 50 times a second part of the operating system, and is transparent to the user. The fact that the operating system does discretize this time, in a minor steps, so that the mouse control code is guaranteed to be performed 50 times per second. These time steps less than we can discern with our eyes (and hands and brain), giving the mouse that appears to be instantaneous and continuous movement, in fact implemented at discrete intervals.

Of course, we are all accustomed to such systems. The cameras at a football game capture frames every 1/25 second, which are transmitted to the TV in the house. Because the time step is so small, it seems that the players on the field are moving in the normal continuous mode, we would expect to see if we were sitting on the bench in the match.

The same situation occurs in the simulation flight. On the plane, the pilot moves a steering wheel. Assuming that a direct cable connection to the control surfaces (and not paying attention to the inertia of the control surface), the elevator moves at once, causing a disturbance to the plane, which is considered as a change in the pitch of the pilot, which reacts with the other control column movement for the correction of the pitch angle. In flight simulator, the position of the handle is digitized, elevator deflection is calculated, the new pitch ratio is computed and the image displayed on the screen via the visual system with a new pitch, allowing the pilot to adjust the attitude of the aircraft. The important point is that the total time for this calculation should be sufﬁciently short, so it seems the pilot instantly. In today's simulator, these calculations should be completed within 1/50-th of second or 20 milliseconds.

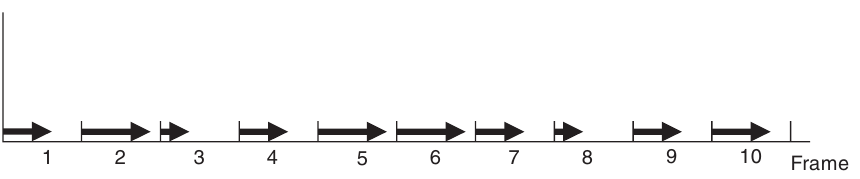
This concept is illustrated in Figure 1.1, which depicts 10 frames of a simulation. The arrow for each frame shows the proportions of the frame used in the calculation of the simulation. If the frame time is sufficiently small, say 1 / 50th of a second, and if any frame in the computation will never exceed the time frame, the real-time simulation.

Fig. 1.1. Real-time frames.

Note that there is an important distinction between the real-time calculations and fast computers. Although real-time simulations may need a fast computer, all calculations have to be completed within the time frame, while for the fast computational problem, only metric is the overall time. Some simulation packages assume that the code generation in a compact form allows the real-time computation. While this may be true for certain applications, in all examples described in this book, the term real-time is used to indicate that all calculations are resolved within one frame for each simulation frame.

Another important factor has been affected. In addition to the basic modeling tasks, the processor may also be required to perform other background tasks. If this additional load calculation results in contradictory within a frame period to perform this task of modeling, simulation in real time cannot be sustained. Therefore, the operating system is an important part of any real-time simulator. The operating system must ensure that it will perform the task of simulating each frame and never introduce delays that cause the problem of modeling a frame exceed its limit.

The safety critical real-time environment, it is necessary to demonstrate that the frame rate in real time can never be broken beyond any reasonable level of doubt. Although the software for flight simulation is not critical for safety, real-time limit still to be performed mode. As a rule, it is the duty of the simulator of the designer. You can monitor the performance of real-time flight simulator and record any violations of the frame period. However, if the frame rate does fall, it is usually apparent to flight crew, as it is noticeable discontinuities in the visual system, or discernible lag in response to the aircraft or even changes in the frequency of the sound output system.

The simulator designer has, in fact, the time budget for the completion of all calculations in the frame and, therefore, trying to use as much of the frame time as possible, as it may leave a small margin for error in these estimates (or for future expansion), in particular as some calculation times data dependent. Taking into account all restrictions on the content of the visual system of the scene, processing flight model, engine model, the weather and so on. It is not uncommon for a frame period should be exceeded sometimes even for full flight simulator, in particular, as the simulator manufacturer cannot have full control over the behavior of the video card in all conditions of flight. However, providing real-time operation, especially for the worst conditions, it is an integral part of validation and acceptance testing system.

### Training versus Simulation.

Flight simulators are used in flight training and is easy to assume that the two terms are synonymous and interchangeable. However, light training provided fulfil l education requirement. The equipment used for this preparation may also include a light simulator, but in this role, just equipment simulator. Simulator, together with an instructor and curriculum is a training package. Confusion has arisen because in some cases, the flight simulator is a pure replacement of the aircraft, and this may have led in the past to purchase flight simulators that were poorly adapted to the requirements of vocational training and is therefore provided a low education.

The first phase of purchases for any flight training program is an analysis of training needs. This establishes that the training program is required, and what is required. On one hand, if the training equipment low, it cannot provide effective training. On the other hand, if the inflated preparation equipment, cost simulation equipment may be excessive.

The often-quoted example of an effective coach is a "cardboard bomber". In the early stages of flight, pilots have to learn a series of checks, including checks before the flight, in-flight inspections, and emergency shutdown test. Sitting in a normal chair, in front of a cardboard facsimile of the cabin, with no moving parts whatsoever, the pilot can indicate on the tool, or press the switch that corresponds to each inspection. Photos used in such a way that each item physically resembles the actual equipment, and is located approximately in the same place as in an airplane. However, the requirement of training equipment just to help pilots remember checks. It is obvious that such equipment would be inappropriate in the later stages of learning, which need real tools aircraft or switches. Similarly, the actual use of aircraft equipment would be an unnecessary expense at the initial stage of training.

For the core curriculum, analysis of training will be conducted by a team of professionals with a good understanding of flying training, simulation technology, teaching methods and human factors. The team will not include flight instructors, which will give further training or simulator company that will produce the equipment. For the airline or military training, specific training will be ed in terms of the desired results (or output) training. For example, the tool for the coach, the requirement may declare "the completion of the training, the pilot must be able to demonstrate ILS approach with a cross wind to the maximum permissible value for the aircraft with one engine failed," as part of the training requirement. The purpose of this single statement is that the simulator requires ILS instrument, realistic model of an engine failure, the model forecast, which includes cross-winds and presumably a database of navigation aids including ILS beacons. Please note that the requirement does not include the elements of motor control, the details of weather patterns or the maximum number of ILS approaches must be provided.

Presumably, such information would be given elsewhere in the training requirement. The training needs analysis team will review the training syllabus, discuss the detailed requirement with the customer (possibly to modify or clarify the requirement), review the technical options (to establish any constraints or parameters) and discuss the training methods used by the instructors. For example, some of the questions to be asked about the ILS simulation might include the following:

* The need for aural cues;
* The accuracy of the simulated ILS;
* The range of the ILS;
* The failure modes of the ILS;
* The method of selecting an ILS frequency;
* The number of ILS channels;
* The failure modes that can be set by an instructor;
* The physical representation of the ILS.

This ﬁnal consideration might give the option of using a graphics display to represent the ILS instrument rather than a mechanical emulation, leading to the clariﬁcation of further issues, including

* The resolution of the ILS display;
* The need for anti-aliasing;
* The update rate of the display.

The important point to bear in mind is that the training requirement extends across the whole scope of the training programme, and even includes non-functional requirements, such as

* Access dimensions;
* Power requirements;
* Emergency lighting;
* Reliability and availability ﬁgures;
* Air-conditioning;
* Safety issues.

Once the training requirements are clearly deﬁned, these are passed to prospective manufacturers who will be invited to tender to supply the training equipment. There is often some variability in these requirements. For example, the requirement may simply state the tasks to be trained and the level of skill to be attained, and possibly the time available to attain it, using the training equipment. In this case, the simulator manufacturer can match their equipment to the training requirement, advocating one technology rather than another.

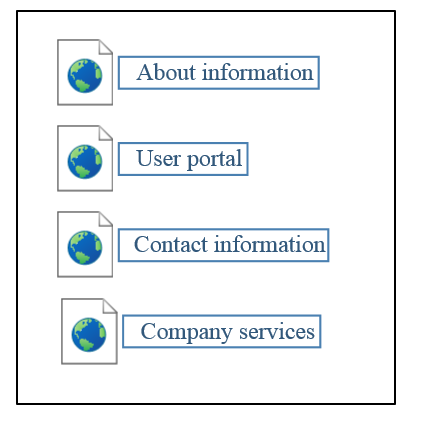
One further issue covered in a training requirement is the method of acceptance. For an airline, their senior pilots are likely to retain close links with the simulator company and go through a formal series of acceptance tests at the factory, prior to shipment and delivery of the simulator, followed by further acceptance tests following the installation.

## Analysis of existing working examples of aircraft and pilot portals.

The idea of portals are to provide brief or full information of products and services that each company provides. Web applications are wide used to provide current needs. Companies sites that have been investigated:

* Panam flight academy.
* Cardif aviation training center.
* FSC training company.

In some cases portals or sites provides full stack of services like user/pilot portal with user data stored and assigned to one. Custom approach is to make a sum of all possible services in one web application: contact information, company description, company services, user portal (Fig. 1.2, Fig. 1.3, Fig. 1.4).



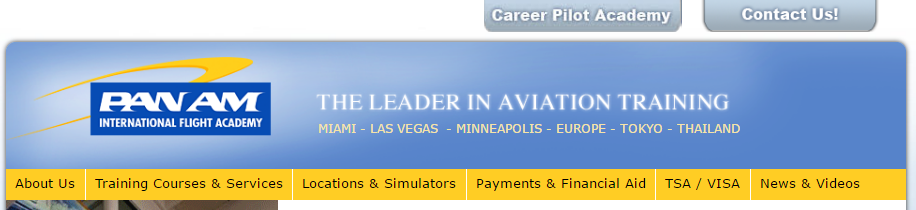


Fig. 1.3. Panam flight academy site.

Fig. 1.2. Common company site map.

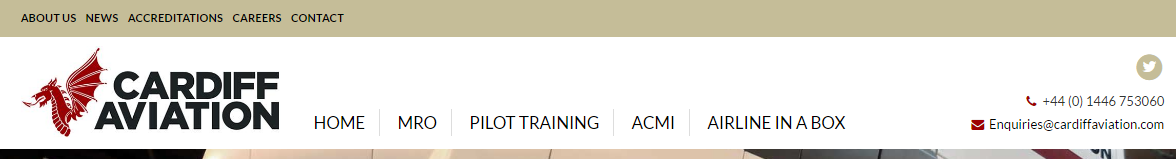
From other side, some company sites and portals provides only service for get-to-know and get familiar with products that company provides (Fig. 1.5). User cannot log in or register to become a member or get member rights on the site. The user/pilot portal is located on internal servers and provides functionality on closed basis.

Fig. 1.4. Cardif aviation training center site.

Due to impossibility of getting access to closed part of user/pilot usability part, the investigation cannot be proceed and only theoretical and logical results is used.

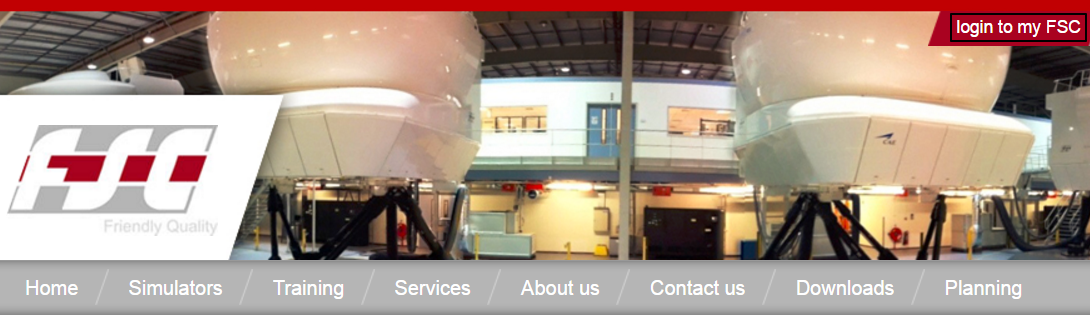


Fig. 1.5. FSC site with log in functionality.

## Client–server architecture.

Client-server model is most used application structure these days. Due to speed of data transition nowadays, this architecture is most effective and that is why it has so much usage. The client–server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters, called clients. Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server host runs one or more server programs, which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers, which await incoming requests.

### Client and server roles.

The client-server characteristic describes the relationship of cooperating programs in an application. The server component provides a function or service to one or many clients, which initiate requests for such services.

Servers are classified by the services they provide. For example, a web server serves web pages and a file server serves computer files. A shared resource may be any of the server computer's software and electronic components, from programs and data to processors and storage devices. The sharing of resources of a server constitutes a service.

Whether a computer is a client, a server, or both, is determined by the nature of the application that requires the service functions. For example, a single computer can run web server and file server software at the same time to serve different data to clients making different kinds of requests. Client software can also communicate with server software within the same computer. Communication between servers, such as to synchronize data, is sometimes called inter-server or server-to-server communication.

### Client and server communication.

In general, a service is an abstraction of computer resources and a client does not have to be concerned with how the server performs while fulfilling the request and delivering the response. The client only has to understand the response based on the well-known application protocol, i.e. the content and the formatting of the data for the requested service.

Clients and servers exchange messages in a request–response messaging pattern. The client sends a request, and the server returns a response. This exchange of messages is an example of inter-process communication. To communicate, the computers must have a common language, and they must follow rules so that both the client and the server know what to expect. The language and rules of communication are defined in a communications protocol. All client-server protocols operate in the application layer. The application layer protocol defines the basic patterns of the dialogue. To formalize the data exchange even further, the server may implement an application programming interface (API). The API is an abstraction layer for accessing a service. By restricting communication to a specific content format, it facilitates parsing. By abstracting access, it facilitates cross-platform data exchange.

A server may receive requests from many distinct clients in a short period. A computer can only perform a limited number of tasks at any moment, and relies on a scheduling system to prioritize incoming requests from clients to accommodate them. To prevent abuse and maximize availability, server software may limit the availability to clients. Denial of service attacks are designed to exploit a server's obligation to process requests by overloading it with excessive request rates.

## Model-view-controller design pattern.

Because of great design value Model–view–controller (MVC) became one of the most usable and popular one in development process of modern web applications. It is a software design pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user. Traditionally MVC was used for desktop graphical user interfaces (GUIs), this architecture has become popular for designing web applications and even mobile, desktop and other clients.

Note that the separation of the model and the UI delegate in the MVC design is extremely advantageous. One unique aspect of the MVC architecture is the ability to tie multiple views to a single model. For example, if you want to display the same data in a pie chart and in a table, you can base the views of two components on a single data model.

### Description of MVC.

As with other software architectures, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system (Fig. 1.6). Particular MVC architectures can vary significantly from the traditional description here.

Fig. 1.6. A typical collaboration of the MVC components.

The central component of MVC, the model, captures the behavior of the application in terms of its problem domain, independent of the user interface.

1. The model directly manages the data, logic, and rules of the application.
2. A view can be any output representation of information, such as a chart or a diagram. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants.
3. The third part, the controller, accepts input and converts it to commands for the model or view.

In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them.

* + 1. A model stores data that is retrieved according to commands from the controller and displayed in the view.
    2. A view generates new output to the user based on changes in the model.
    3. A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., scrolling through a document).

### Use in web applications

Although originally developed for desktop computing, MVC has been widely adopted as an architecture for World Wide Web applications in major programming languages. Several commercial and noncommercial web frameworks have been created that enforce the pattern. These software frameworks vary in their interpretations, mainly in the way that the MVC responsibilities are divided between the client and server.

Early web MVC frameworks took a thin client approach that placed almost the entire model, view and controller logic on the server. This is still reflected in popular frameworks such as PHP, Django, Rails and ASP.NET MVC. In this approach, the client sends either hyperlink requests or form input to the controller and then receives a complete and updated web page (or other document) from the view; the model exists entirely on the server. As client technologies have matured, frameworks such as AngularJS, EmberJS, JavaScriptMVC and Backbone have been created that allow the MVC components to execute partly on the client (also see Ajax).

## Analysis of examples software tools and technologies for architecture of aircraft companies sites and portals.

Due to investigation process, it has been found out that simple architectural patterns and models are commonly used. The usage of not deep models provides easiness of understanding and development process. Because of complicated business logic is redundant in portal’s functionality it is effective to use less demanded to resources models. Common use of simple architectural decisions caused by effective spent of resources that are need to create fully functional software product. For instance, portals that have been analyzes above provide only basic business logic like information display.

From other side, complicated business solutions with inner functionality like authentication and authorization need to implement more reliable and complicated models or patterns. Security is important part of these systems. The implementation of hard-maintained functions, or functionality that cannot be monitored, demands usage of solutions, methods or even external products that are costly and difficult to get to know and even use.

Complex solutions can be got, bought or even developed by developers that are creating product. Free products are wide spread in a sphere of development web applications, even provided by owner of some enterprise package for some languages. The example is Java. Oracle provides Java Enterprise. Java Enterprise Edition or Java EE is a widely used computing platform for enterprise software. The platform provides an API runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications.

### Session management.

One huge variety of functionality and possibilities that JEE provides is session management. This is mostly used for authentication on web application.

Session is a conversional state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response.

In Java EE HttpServelet interface is used. A Java servlet is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user. A session usually corresponds to one user, who may visit a site many times. The server can maintain a session in many ways such as using cookies or rewriting URLs.

This interface allows servlets to:

1. View and manipulate information about a session, such as the session identifier, creation time, and last accessed time
2. Bind objects to sessions, allowing user information to persist across multiple user connections

When an application stores an object in or removes an object from a session, the session checks whether the object implements HttpSessionBindingListener. If it does, the servlet notifies the object that it has been bound to or unbound from the session. Notifications are sent after the binding methods complete. For session that are invalidated or expire, notifications are sent after the session has been invalidated or expired.

### Java Servlet.

A Java servlet is a Java program that extends the capabilities of a server. Although servlets can respond to any types of requests, they most commonly implement applications hosted on Web servers. Such Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.

Servlets are most often used to process or store a Java class in Java EE that conforms to the Java Servlet API, a standard for implementing Java classes that respond to requests. Servlets could in principle communicate over any client–server protocol, but they are most often used with the HTTP protocol. Thus, "servlet" is often used as shorthand for "HTTP servlet". Thus, a software developer may use a servlet to add dynamic content to a web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML. Servlets can maintain state in session variables across many server transactions by using HTTP cookies, or rewriting URLs.

To deploy and run a servlet, a web container must be used. A web container (also known as a servlet container) is essentially the component of a web server that interacts with the servlets. The web container is responsible for managing the lifecycle of servlets, mapping a URL to a particular servlet and ensuring that the URL requester has the correct access rights.

The Servlet API, contained in the Java package hierarchy javax.servlet, defines the expected interactions of the web container and a servlet.

A Servlet is an object that receives a request and generates a response based on that request. The basic Servlet package defines Java objects to represent servlet requests and responses, as well as objects to reflect the servlet's configuration parameters and execution environment. The package javax.servlet.http defines HTTP-specific subclasses of the generic servlet elements, including session management objects that track multiple requests and responses between the web server and a client. Servlets may be packaged in a WAR file as a web application (Fig.1.7).

Servlets can be generated automatically from Java Server Pages (JSP) by the JavaServer Pages compiler. The difference between servlets and JSP is that servlets typically embed HTML inside Java code, while JSPs embed Java code in HTML. While the direct usage of servlets to generate HTML (as shown in the example below) has become rare, the higher level MVC web framework in Java EE (JSF) still explicitly uses the servlet technology for the low level request/response handling via the FacesServlet. A somewhat older usage is to use servlets in conjunction with JSPs in a pattern called "Model 2", which is a flavor of the model–view–controller.

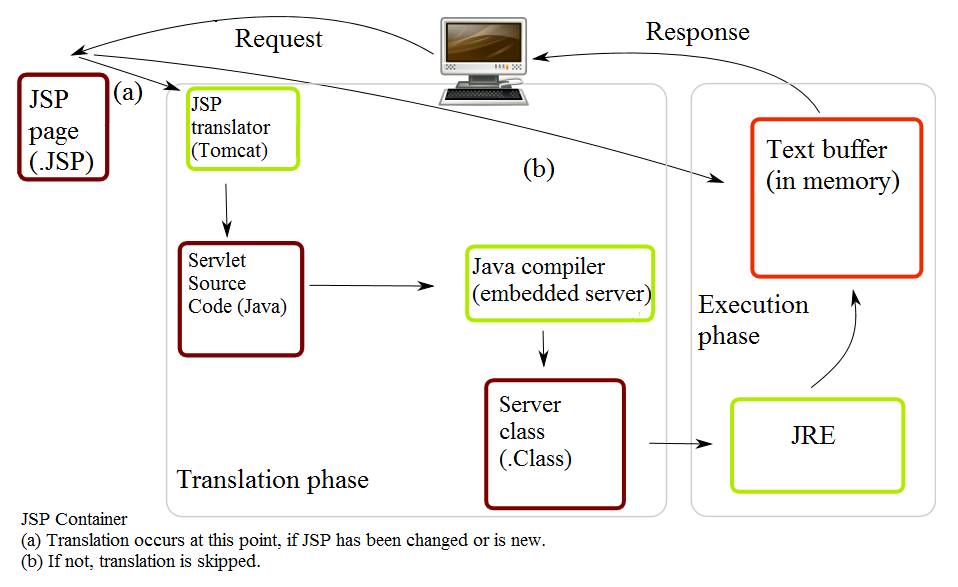
The current version of Servlet is 3.1.

Fig.1.7. Life of a JSP file.

## The evaluation and maintain the skills of pilots.

Huge aircraft companies are usually use software developed only for internal usage. This ensure security for software, users and company itself. These systems are unavailable for common user, the access can be granted only for administrators, pilots or supervisors. Nevertheless, the logic of these systems is understandable. Main idea is to process, calculate and perform evaluation of pilots with using gathering data from flight simulator and calculation results of training with the usage of human, that is usually a supervisor or someone who know the standards of flight simulation. From other side it can be system with implemented formula or formulas for calculation and evaluation metrics gathered from flight simulation.

The number of evaluation methodologies is equal to number of flight simulators. There are evaluation systems that evaluate pilot synergy and communication during flights and specific situations. For example CrewFactors created by Vocavio. It measures excellence in team communication skills to enhance safety and operational systems.

Other one is Fit for Work Service(FFWS) invented and implemented by Department for Work and Pensions. to offer support for people in the early stages of sickness absence, particularly for employees working in small and medium-sized enterprises (SMEs). It was envisaged that case-managed and multidisciplinary services would provide a personalized help to address both social concerns, such as financial and housing issues, and clinical needs, and as a consequence would keep people in work. Between April and June 2010, FFWS pilots were launched in 11 areas throughout Great Britain with the intention of testing different approaches to providing the service, and getting people back to work as quickly as possible. Pilots were formed by partnerships of health, employment and local community organizations, and offered bio psychosocial assessments of need and case-managed support to aid a quick return to work. From April 2011, seven of the pilots were funded for up to a further two years.

Given examples of evaluation methodologies or systems just represent the variability and the totality of ones in different aspects of evaluation aircraft staff. Nevertheless, one of or maybe the most necessary set of skills is to perform systematic, effective, logical, safe procedures that are required in the process of controlling plains.

## Flight Skill Decay with Non-practice

Early research examining the loss or decay of pilot flight skills used crude flight simulators, or suspended aircraft models. This initial research focused on assessing the recall ability of previously trained skills after a time of disuse, and found that proficiency declines after a period of non-practice (Ammons, Farr, Bloch, Neumann, Dey, Marion, & Ammons, 1958; Fleishman & Parker, 1962; Wright, 1973). There was found that the decay of flight skills was present regardless of the duration of elapsed time without practice. Participants were given up to eight hours of training to proficiency for a simulated flight task.

After a “no-practice interval” from 24 hours to two years, a greater loss of skill occurred as time since the last practice increased. Flight skill quickly returned to proficiency, up to 75 percent, in as little as five minutes of practice after the hiatus. Certified pilots also suffered from “profound…rapid… and pervasive” (Childs, Spears, & Prophet, 1983, p. 30) flight skill loss after relatively short periods of non-practice. Private pilots who did not continuously practice flight maneuvers, especially those critical during aircraft emergencies, would quickly lose proficiency in the procedure or the application of those maneuvers in as little as eight months.

In the case of Colgan Air Flight 3407, when the Captain recognized the aircraft was in an aerodynamic stall, he incorrectly applied the required technique for recovery, exacerbating the condition, and rendered the aircraft unrecoverable (NTSB, 2010). Investigators were unable to determine why a certified Captain would act inappropriately to a flight maneuver that is evaluated during initial and recurrent training. Typically, Captains are required to successfully demonstrate these maneuvers every six months while First Officers receive this training once a year.

The training is intended to maintain the proficiency of flight crews in identifying and reacting appropriately to in-flight emergencies. The flying environment today has changed to that of less manual flying and more use of automation. Furthermore, the type of operation also dictates the amount of practice a pilot receives. The shorter trips flown by domestic carriers offer both pilots a daily opportunity to practice their skills. However, that is in sharp contrast to international pilots who may only get a chance to operate the controls a few times per year. Relief pilots during international flights rotate positions to allow the Captain and First Officer an opportunity to rest during cruise flight and normally do not get an opportunity to actually manipulate the controls.

The lack of actual flying experience from international flight crews may have contributed to a Sydney bound United Airlines flight that came within 100 feet of a mountain after takeoff from San Francisco in 1999. After experiencing an engine failure, the flying pilot of the B-747-400 did not perform the proper recovery technique, which exacerbated the critical condition of the aircraft and nearly collided with a mountain. The one takeoff and landing the pilot had performed the week before the incident was the first in nearly a year.

### Interaction pilots with simulator desk.

A key issue for enhancing flight operations safety is to support more effectively the interaction between flight crews and flight deck automation (specifically, autopilot, auto throttle, and the Flight Management Computer). The introduction of automation to the “glass cockpit” has provided numerous benefits, such as increased precision and efficiency. However, these benefits occur primarily in situations where the automation performs tasks that don’t require pilot involvement. In circumstances that require cooperation and coordination between pilots and automated systems, unexpected problems are being encountered.

Numerous recent studies have demonstrated that pilots can become confused about the state and/or behavior of flight deck automation. One consequence of breakdowns in pilot-automation coordination is the pilot’s loss of mode awareness. Mode awareness refers to the knowledge and understanding of the current and future state and behavior of the automation. This loss of mode awareness can lead to mode errors and automation surprises. Mode errors, generally speaking, occur when a pilot performs an action appropriate for the assumed system state but not for the actual state. Or, a mode error can refer to the omission of a required action or intervention with automation actions. Mode errors lead to automation surprises when the pilot notices that the automation is engaged in activities that were not commanded (or, fails to engage in activities that were thought to be commanded). Both mode errors and automation surprises have played a role in recent incidents and accidents and can lead to poor or slow compliance with ATC clearances (deviations from assigned altitudes).

Several factors can contribute to a loss of mode awareness:

* the pilot can have an incomplete and/or inaccurate mental model of the flight deck automation;
* the automation feedback can be inadequate because it fails to support pilots in predicting, assessing, and understanding current system state and behaviors;
* the highly complex logic underlying flight deck automation behavior that differs from pilots’ reasoning about their flying tasks, and differs considerably across manufacturers, aircraft types, and in some cases, across individual planes within type (due to software upgrades).

One avenue for removing these problems is to modify the flight deck interface to increase the salience of changes that can occur without explicit pilot commands. In addition, pilots need better support for interpreting the indicated automation state in terms of its implications for current and future aircraft behavior. New flight deck interface designs are being developed with these requirements in mind. However, design changes take considerable time to find their way into the fleet, and solutions are needed for the existing fleet. Therefore, new approaches to automation training also need to be developed and implemented. In fact, efforts are currently underway to enhance pilot training and improve pilot mental models of the flight deck automation.

A second approach to addressing mode confusion relates to the fact that pilots are not well supported in learning how to monitor automation-related indications effectively. The “accepted wisdom” on scanning cockpit indications for years was based on the “T” pattern of primary indications (airspeed, attitude, altitude, and heading). However, with the advent of integrated flight deck displays and highly complex automated systems on glass cockpit aircraft, the pilot needs to monitor a larger, more diverse, and more distributed set of indications. Although the primary flight display (PFD) and the navigation (Nav) display integrate most of the primary indications, the pilot also needs to monitor the mode control panel (MCP) and the flight management computer (FMC), which is accessed through the control data unit (CDU). There are no documented strategies for effectively monitoring this diverse set of indications, and, as a result, pilots often develop their own—not necessarily effective— approaches to the task.

### Cognitive Aspects of Flight

Flight does not exclusively involve motor skill but is also highly dependent upon cognitive processing, which is just as susceptible to decay after periods of. Flying is a psychomotor process, involving both motor skills and cognitive processing to achieve the desired flight path and maintain adequate situational awareness. In 1986 was found that the majority of flying skill was attributed to cognitive performance and proficiency. Flight by reference to instruments, placed significant cognitive demands on pilot participants, and revealed that this type of flying was most affected after nonpractice intervals. Recent research has also revealed that cognitive skills, in addition to physical skills, decrease over time without proper practice, especially those skills that were learned early in training but not used for extended period.

Cognitive processing is a crucial skill involved in nearly every aspect of piloting. Visual and other sensory cues, combined with flight data, all must be efficiently processed for the pilot to make adequate and appropriate inputs to control the aircraft as desired. For example, small corrections are made to the flight controls, based on information from the flight instruments, to track a desired course or maintain a specified altitude. The sensing of flight data is an interpretation and processing, and subsequent physical adjustments of the flight control to achieve the desired flight outcome, as being a “closed-loop” control task. Pilots who are manually flying are continuously performing this closed-loop processing. This skill is fundamental in the accurate monitoring of an aircraft’s progress along a route of flight. Closed-loop processing is the most demanding cognitive process performed on the flight deck because so much information must be understood and acted upon in a very short period of time.

There was found that pilots who had significant experience flying traditional, non-glass cockpit aircraft, developed robust mental models of performance characteristics during different phases of flight. These heuristics allowed experienced pilots to quickly and accurately predict and anticipate exactly how the aircraft would perform, thus reducing the high processing demands imposed by closed-loop processing. These pilots developed their own schema for the operation of the aircraft based upon experience with power settings, descent profiles, and rules of thumb. They no longer had to perform complex mathematical calculations to determine when to begin a descent; rather they could simply apply the heuristic model for that situation. Less experienced pilots, lack these heuristics and quickly become saturated, resulting in poor aircraft control and planning. Over-dependence on automated systems exacerbates this issue and further inhibits the ability to develop the required mental models for manual flight.

Was conducted research on manual flight skill of pilots transitioning from light twin engine training aircraft to modern airliners. By testing their performance both before and after a 40-hour jet transition course, the differences in control strategies became apparent. The students did not have the proper experience to develop the schema needed to understand how the aircraft would react to different power and pitch settings. The result was large, coarse control inputs to achieve a desired aircraft condition. The students also had significant difficulty in managing the inertia and energy of the larger aircraft, and therefore had more trouble in predicting where in space the aircraft would arrive at a given period of time. When measured after the 40-hour training course, student performance improved most notably in their ability to anticipate the performance of the jet aircraft and make smooth and precise control inputs for the desired outcome.

## Approach for pilot evaluation.

Purpose of comprehensive flight simulators is to perform training and professional evaluation of aircraft. The process of training is determined by the program, which include:

* Basic information about terms and conditions of training places;
* A list of exercises to perform crew;
* Criteria for assessing the knowledge and skills;
* Rules of registration documentation training;
* A description of procedures for exercise.

Description of exercise procedures include:

* Flight conditions;
* Main circuit exercise (eg, takeoff, climb, fly in the zone);
* Failures and situations practiced in the exercise;
* Temporary regulation exercises.

The program does not formalize the specific indicators and values of individual knowledge and skills assessments, limited to general guidelines for the assessment and the rules of formation of an overall assessment for the pilots. It is understood that the instructors get the scores on the basis of subjective criteria known to them.

To automate the evaluation is necessary to generate a set of measured values that will provide estimates of the value of knowledge and skills. To do this, try to use a known technique "goal-question-metric" in the field of software engineering.

Conceptual level (Goal) - A goal is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view and relative to a particular environment.

Operational level (Question) - A set of questions is used to define models of the object of study and then focuses on that object to characterize the assessment or achievement of a specific goal.

Quantitative level (Metric) - A set of metrics, based on the models, is associated with every question in order to answer it in a measurable way.

Table 1.1

Goal-question-metric table of pilot evaluation

|  |  |  |
| --- | --- | --- |
| Goal | Question | Metric |
| Assessment of correctness parry the failure of one engine. | Was screw of failed engine been vaned? | Indicator of vaning has become equal to 1 after the rejection. |

Goal-question-metric table of pilot evaluation

|  |  |  |
| --- | --- | --- |
| Goal | Question | Metric |
| Assessment of correctness parry the failure of one engine. | Has failed engine been turned off? | Fuel valve closed |
|  |  | The stop valve has been closed |
|  | Was there a difference of traction trimmer aileron compensated? | The bank is less than 5 degrees in 30 seconds. |
|  | Was there a difference compensated with thrust rudder trim? | Yaw is less than 5 degrees in 30 seconds. |
| Rate the quality of touch the runway | Was it a soft touch? | Vertical speed at the moment of contact. |
|  |  | The horizontal speed at the moment of contact. |
|  | How right was the position of the aircraft at the moment of contact? | <Angle of list> |
|  |  | < Pitch> |
|  |  | < Speed angle relative to the axis of runway < |
|  | What was the runway touchdown accuracy? | The deviation from the optimal longitudinal line of contact. |
|  |  | The deviation from the optimum transverse line of contact |

Goal-question-metric table of pilot evaluation

|  |  |  |
| --- | --- | --- |
| Goal | Question | Metric |
| Rate the quality of braking | As far exceeded the braking distance? | The difference between the actual and maximum allowable distance. |
|  | What is a departure from the runway axis during braking? | The standard deviation from the runway axis. |
|  | Do all measures have been taken for braking? | The value of the brake pedal |
|  |  | Turn on the reverse |
|  |  | Enabled interceptors |

## Conclusion

The result of current investigation process it was received that flight simulators, systems of control and sub-systems for user usage represent whole ecosystem of service called pilot simulator trainings.

From perspective of simulator system it is synergy of electronics and software that corresponds to needs, desires of pilots, instructors and companies that buy them, and companies that produce them. But if speaking easier, simulator is just a toy in hands of pilot. Evaluation systems open true purpose of simulators. They help pilots or instructors get to know almost anything about skills of pilot, metrics of test flight and even evaluate pilot with the usage of implemented calculation methods.

From other side are portals and introduction sites that help unknown companies, which produce service like testing and training for pilots, gather more users and big companies to inform their users about something new.

This kind of sum of sub-system in one main system is complicated to develop. Each sub-system has each own purpose and goals, each hast to work differently and give different results. The resources that are spent on each are different too. That is why building the whole, lets say, eco-system of training environment is pretty ineffective. The better approach is to divide development of each. Due to this training center can vary elements of this eco-system to achieve what is necessary: flight simulator, system of evaluation of communication of staff, evaluation environment and user portal, where pilots can monitor information about their skills and progresses.

Approach of evaluation of pilots has been described. Listed three danger situations with related metrics that hast to be checked for evaluation. More situations can be implemented in the process of expanding functionality. This requires the pilot to be aware of a number of factors enabling him to assess the situation and keep clear from any danger to the flight.

# Chapter 2.

# analyzis and REQUIREMENTS SPECIFICATION.

## 2.1 Overview of system structure.

System covers almost all necessary areas of user services and provide full maintenance of these services. The whole system will be structured of four sub-systems that are described in Table 2.1.

Table 2.1.

Sub-systems description and purposes.

|  |  |
| --- | --- |
| Sub-system | Purpose |
| Flight simulator | Flight simulation is used to perform gathering data of pilot training process. |
| Main web-application | Web application stored locally and perform gathering data, calculation and user interface for comfort usage of application in web browser. Perform writing data and instructor evaluation to database. |
| User portal | Web site or portal for users to get to know the information about information center and evaluation of pilots. |
| Database | Storing the data for both main application and user portal. |

As seen from table all subsystems will be interconnected and working with database but in different ways.

### 2.1.2 Functional requirements.

Main application functional requirements:

1. Ability to receive data from flight simulator.
2. Ability to process data from flight simulator.
3. Ability to expand data with additional information (username, plain model, etc.).
4. Ability to access database with read-write rights.
5. Ability to manage user accounts.
6. Ability to manage plain models.
7. Ability to manage metrics.
8. Ability to manage flight situations.
9. Ability to set metrics to flight situation.
10. Ability to manage flight simulation records.
11. Ability to evaluate flight simulation records.
12. Ability to connect to flight simulator with socket.
13. Ability of authentication with session.
14. Ability of authorization with session.
15. Ability to authorize with administrator rights.
16. Ability for administrator to set roles.
17. Ability to manage data in database.
18. Ability of session management.
19. Ability to receive data from database.
20. Ability to visualize data received from database.
21. Ability to connect to application from only from local computers.
22. Ability to deploy application on different operational systems.
23. Ability to use application functionality with browser.
24. Ability to start session of gathering data from simulation.
25. Ability to stop session of gathering data from simulation.
26. Ability to setup port with which simulator can send data to application.

User portal functional requirements:

1. Ability to access database with read-only rights.
2. Ability to receive data from database.
3. Ability to visualize data received from database.
4. Ability of authentication with session.
5. Ability of authorization with session.
6. Ability to authorize with administrator rights.
7. Ability to deploy application on different operational systems.
8. Ability to use portal/site with browser.
9. Ability to perform visualization of possible data of flight simulator center.
10. Ability to perform easy visualization of record data gathered from database.

## 2.2 System architecture overview

Both main application and user portal use client-server architecture.

Client-server architecture is one of the architectural templates software and is the dominant concept in the creation of distributed network applications and provides for cooperation and the exchange of data between them. It provides the following key components:

• a set of servers that provide information services or other programs that appeal to them;

• set of clients using services provided by servers;

• network that provides interaction between clients and servers.

Servers are independent of each other. Customers also operate in parallel and independently of each other. No strict binding clients to servers. More than a typical situation is when one server simultaneously handles requests from different clients; on the other hand, the client can then apply to a single server, then to another. Customers should know about available servers, but may not have any idea about the existence of other customers.

### 2.2.1 Interface of system.

Interface of main application provides following capabilities:

* 1. User should login with credentials he/she has been provided.
  2. User can use menu items to browse through application.
  3. On main page administrator or instructor can setup port and start session of gathering data.
  4. On main page administrator or instructor can stop session.
  5. User can look at all records that have been saved in system due to his access rights.
  6. User can get all plain models.
  7. User can get all flight situations.
  8. User can get all metrics from selected flight situations.
  9. User can log out from main application.
  10. Administrator can manage users’ accounts.
  11. Administrator can manage plain models.
  12. Administrator can manage flight situations.
  13. Administrator can manage metrics from selected flight situation.
  14. Administrator can manage records from flight simulator.

Interface of user portal provides following capabilities:

1. User can get to know information about flight simulation center.
2. User can log in and look through his last flight simulation records.
3. User can browse thought site with menu.
4. User can get all plain models.
5. User can get all flight situations.

### 2.3 Instruments for development of software.

Both user portal and main application will be using same technologies and instruments for development. This makes development process less complicated and more maintained. The architecture is client-server and main design pattern is Mode-View-Controller (MVC).

### 2.3.1 Model–view–controller

Model–view–controller (MVC) is a software design pattern for implementing user interfaces on computers. It divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

### 2.3.2 Separation of concerns.

In computer science, separation of concerns (SoC) is a design principle for separating a computer program into distinct sections, such that each section addresses a separate concern. A concern is a set of information that affects the code of a computer program. A concern can be as general as the details of the hardware the code is being optimized for, or as specific as the name of a class to instantiate. A program that embodies SoC well is called a modular program. Modularity, and hence separation of concerns, is achieved by encapsulating information inside a section of code that has a well-defined interface. Encapsulation is a means of information hiding. Layered designs in information systems are another embodiment of separation of concerns (e.g., presentation layer, business logic layer, data access layer, persistence layer).

In software engineering, the terms front end and back end refers to the separation of concerns between the presentation layer (front end), and the data access layer (back end) of a piece of software, or the physical infrastructure or hardware.

In software design, the model-view-controller architecture provides front and back ends for the database, the user and the data processing components. The "model" and "controller" make up the back, while the "view" makes up the front.

In content management systems, the terms front end and back end may refer to the end-user facing views of the CMS and the administrative views, respectively.

In speech synthesis, the front end refers to the part of the synthesis system that converts the input text into a symbolic phonetic representation, and the back end converts the symbolic phonetic representation into actual sounds.

For major computer subsystems, a graphical file manager is a front end to the computer’s file system, and a shell interfaces with the operating system. The front end faces the user, and the back end launches the programs of the operating system in response.

In compilers, the front end translates a computer programming source code into an intermediate representation, and the back end works with the intermediate representation to produce code in a computer output language. The back end usually optimizes to produce code that runs faster. The front-end/back-end distinction can separate the parser section that deals with source code and the back end that generates code and optimizes. Some designs, such as GCC, offer choices between multiple front ends (parsing different source languages) or back ends (generating code for different target processors).

Using the command-line interface (CLI) requires the acquisition of special terminology and memorization of commands, so a graphical user interface (GUI) acts as a front end desktop environment instead.

### 2.3.3 Back-end technologies.

The language for development back-end or data-access level of both applications is Java and Java EE like widely used computing platform for enterprise software. Data management will consist two technologies: Hibernate and JPA.

### 2.3.4 Java Persistence API.

The Java Persistence API (JPA) is a Java application programming interface specification that describes the management of relational data in applications using Java Platform, Standard Edition and Java Platform, Enterprise Edition.

Persistence in this context covers three areas:

* the API itself, defined in the javax.persistence package;
* the Java Persistence Query Language (JPQL);
* object/relational metadata.

The Java Persistence Query Language (JPQL) makes queries against entities stored in a relational database. Queries resemble SQL queries in syntax, but operate against entity objects rather than directly with database tables.

**2.3.5 Hibernate ORM.**

Hibernate is an object-relational mapping tool for the Java programming language. It provides a framework for mapping an object-oriented domain model to a relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct, persistent database accesses with high-level object handling functions.

Hibernate is free software that is distributed under the GNU Lesser General Public License 2.1.

Hibernate's primary feature is mapping from Java classes to database tables, and mapping from Java data types to SQL data types. Hibernate also provides data query and retrieval facilities. It generates SQL calls and relieves the developer from the manual handling and object conversion of the result set.

The mapping of Java classes to database tables is implemented by the configuration of an XML file or by using Java Annotations. When using an XML file, Hibernate can generate skeleton source code for the persistence classes. This is auxiliary when annotations are used. Hibernate can use the XML file or the Java annotations to maintain the database schema.

There are provided facilities to arrange one-to-many and many-to-many relationships between classes. In addition to managing associations between objects, Hibernate can also manage reflexive associations wherein an object has a one-to-many relationship with other instances of the class type.

Hibernate supports the mapping of custom value types. This makes the following scenarios possible:

1. Overriding the default SQL type when mapping a column to a property.
2. Mapping Java Enums to columns as though they were regular properties.
3. Mapping a single property to multiple columns.

Definition: Objects in an object-oriented application follow OOP principles, while objects in the back-end follow database normalization principles, resulting in different representation requirements. This problem is called "object-relational impedance mismatch". Mapping is a way of resolving the object-relational impedance mismatch problem.

Mapping informs the ORM tool of what Java class object to store in which database table.

Hibernate provides transparent persistence for Plain Old Java Objects (POJOs). The only strict requirement for a persistent class is a no-argument constructor, not necessarily public. Proper behavior in some applications also requires special attention to the equals() and hashCode() methods.

Collections of data objects are typically stored in Java collection classes such as implementations of the Set and List interfaces. Java generics, introduced in Java 5, are supported. Hibernate can be configured to lazy load associated collections. Lazy loading is the default as of Hibernate 3.

Related objects can be configured to cascade operations from one to the other. For example, a parent Album object can be configured to cascade its save and/or delete operation to its child Track objects.

In Hibernate jargon, an entity is a stand-alone object in Hibernate's persistent mechanism which can be manipulated independently of other objects. In contrast, a component is subordinate to an entity and can be manipulated only with respect to that entity. For example, an Album object may represent an entity; but the Tracks object associated with the Album objects would represent a component of the Album entity, if it were assumed that Tracks could only be saved or retrieved from the database through the Album object. Unlike J2EE, Hibernate can switch databases.

### 2.3.6 Data access object (DAO).

In computer software, a data access object (DAO) is an object that provides an abstract interface to some type of database or other persistence mechanism. By mapping application calls to the persistence layer, the DAO provides some specific data operations without exposing details of the database. This isolation supports the Single responsibility principle. It separates what data access the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), from how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).

Although this design pattern is equally applicable to the following: (1- most programming languages; 2- most types of software with persistence needs; and 3- most types of databases) it is traditionally associated with Java EE applications and with relational databases.

The advantage of using data access objects is the relatively simple and rigorous separation between two important parts of an application that can but should not know anything of each other, and which can be expected to evolve frequently and independently. Changing business logic can rely on the same DAO interface, while changes to persistence logic do not affect DAO clients as long as the interface remains correctly implemented. All details of storage are hidden from the rest of the application (see information hiding). Thus, possible changes to the persistence mechanism can be implemented by just modifying one DAO implementation while the rest of the application isn't affected. DAOs act as an intermediary between the application and the database. They move data back and forth between objects and database records. Unit testing the code is facilitated by substituting the DAO with a test double in the test, thereby making the tests non-dependent on the persistence layer.

Potential disadvantages of using DAO include leaky abstraction, code duplication, and abstraction inversion. In particular, the abstraction of the DAO as a regular Java object can hide the high cost of each database access, and can also force developers to trigger multiple database queries to retrieve information that could otherwise be returned in a single operation with normal SQL set operations. If an application requires multiple DAOs, one might find oneself repeating the essentially the same create, read, update, and delete code for each DAO. This boiler-plate code may be avoided however, by implementing a generic DAO that handles these common operations. Time consumption is moderate.

### 2.3.7 Java API for RESTful Web Services.

JAX-RS: Java API for RESTful Web Services (JAX-RS) is a Java programming language API spec that provides support in creating web services according to the Representational State Transfer (REST) architectural pattern. JAX-RS uses annotations, introduced in Java SE 5, to simplify the development and deployment of web service clients and endpoints.

From version 1.1 on, JAX-RS is an official part of Java EE 6. A notable feature of being an official part of Java EE is that no configuration is necessary to start using JAX-RS. For non-Java EE 6 environments a (small) entry in the web.xml deployment descriptor is required.

JAX-RS provides some annotations to aid in mapping a resource class (a POJO) as a web resource. The annotations include:

@Path specifies the relative path for a resource class or method.

@GET, @PUT, @POST, @DELETE and @HEAD specify the HTTP request type of a resource.

@Produces specifies the response Internet media types (used for content negotiation).

@Consumes specifies the accepted request Internet media types.

In addition, it provides further annotations to method parameters to pull information out of the request. All the @\*Param annotations take a key of some form which is used to look up the value required.

* @PathParam binds the method parameter to a path segment.
* @QueryParam binds the method parameter to the value of an HTTP query parameter.
* @MatrixParam binds the method parameter to the value of an HTTP matrix parameter.
* @HeaderParam binds the method parameter to an HTTP header value.
* @CookieParam binds the method parameter to a cookie value.
* @FormParam binds the method parameter to a form value.
* @DefaultValue specifies a default value for the above bindings when the key is not found.
* @Context returns the entire context of the object (for example @Context HttpServletRequest request).

### 2.3.8 Front-end technologies.

The basis for front-end part is common for nowadays – HTML5 for markup the page, CSS to stylize page markup and JavaScript for processing the basic calculations on side of user. The JavaScript framework for expanding the possibilities of development is AngularJS. Factory for REST interconnection with back-end is ngResource factory.

### 2.3.9 AngularJS framework.

AngularJS (commonly referred to as "Angular" or "Angular.js") is a complete JavaScript-based open-source front-end web application framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications. The JavaScript components complement Apache Cordova, the framework used for developing cross-platform mobile apps. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–viewmodel (MVVM) architectures, along with components commonly used in rich Internet applications.

The AngularJS framework works by first reading the HTML page, which has embedded into it additional custom tag attributes. Angular interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of those JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources.

AngularJS is built on the belief that declarative programming should be used to create user interfaces and connect software components, while imperative programming is better suited to defining an application's business logic. The framework adapts and extends traditional HTML to present dynamic content through two-way data-binding that allows for the automatic synchronization of models and views. As a result, AngularJS de-emphasizes explicit DOM manipulation with the goal of improving testability and performance.

AngularJS's design goals include:

* + to decouple DOM manipulation from application logic. The difficulty of this is dramatically affected by the way the code is structured.
  + to decouple the client side of an application from the server side. This allows development work to progress in parallel, and allows for reuse of both sides.
  + to provide structure for the journey of building an application: from designing the UI, through writing the business logic, to testing.

Angular implements the MVC pattern to separate presentation, data, and logic components. Using dependency injection, Angular brings traditionally server-side services, such as view-dependent controllers, to client-side web applications. Consequently, much of the burden on the server can be reduced.

Angular uses the term "**scope**" in a manner akin to the fundamentals of computer science.

Scope in computer science describes when in the program a particular binding is valid. The ECMA-262 specification defines scope as: a lexical environment in which a Function object is executed in client-side web scripts; akin to how scope is defined in lambda calculus.

As a part of the "MVC" architecture, the scope forms the "Model", and all variables defined in the scope can be accessed by the "View" as well as the "Controller". The scope behaves as a glue and binds the "View" and the "Controller".

In Angular, "scope" is a certain kind of object that itself can be in scope or out of scope in any given part of the program, following the usual rules of variable scope in JavaScript like any other object. When the term "scope" is used below, it refers to the Angular scope object and not the scope of a name binding. Angular is an easy way to bind data in HTML DOM.

### 2.3.10 Bootstrap

The tasks performed by the AngularJS bootstrapper occur in three phases after the DOM has been loaded:

* Creation of a new Injector
* Compilation of the directives that decorate the DOM
* Linking of all directives to scope

AngularJS directives allow the developer to specify custom and reusable HTML-like elements and attributes that define data bindings and the behavior of presentation components. Some of the most commonly used directives listed in AngularJS directives (Table 2.2).

Bootstrap is modular and consists essentially of a series of Less stylesheets that implement the various components of the toolkit. A stylesheet called "Bootstrap less" includes the components stylesheets. Developers can adapt the Bootstrap file itself, selecting the components they wish to use in their projects.

Adjustments are possible to a limited extent through a central configuration stylesheet. More profound changes are possible by the Less declarations.

The use of the Less stylesheet language allows the use of variables, functions and operators, nested selectors

Bootstrap provides a set of stylesheets that provide basic style definitions for all key HTML components. These provide a uniform, modern appearance for formatting text, tables and form elements.

In addition to the regular HTML elements, Bootstrap contains other commonly used interface elements. The components are implemented as CSS classes, which must be applied to certain HTML elements in a page.

Bootstrap comes with several JavaScript components in the form of jQuery plugins. They provide additional user interface elements such as dialog boxes, tooltips, and carousels. They also extend the functionality of some existing interface elements, including for example an auto-complete function for input fields. In version 2.0, the following JavaScript plugins are supported: Modal, Dropdown, Scrollspy, Tab, Tooltip, Popover, Alert, Button, Collapse, Carousel and Typeahead.

Table 2.2.

AngularJS directives

|  |  |
| --- | --- |
| Directives | Description |
| ng-app | Declares the root element of an AngularJS application, under which directives can be used to declare bindings and define behavior. |
| ng-bind | Sets the text of a DOM element to the value of an expression. For example, <span ng-bind="name"></span> displays the value of ‘name’ inside the span element. Any change to the variable ‘name’ in the application's scope reflect instantly in the DOM. |
| ng-model | Similar to ng-bind, but establishes a two-way data binding between the view and the scope. |
| ng-model-options | Provides tuning for how model updates are done. |
| ng-class | Lets class attributes be dynamically loaded. |
| ng-controller | Specifies a JavaScript controller class that evaluates HTML expressions. |
| ng-repeat | Instantiate an element once per item from a collection. |
| ng-show & ng-hide | Conditionally show or hide an element, depending on the value of a boolean expression. Show and hide is achieved by setting the CSS display style. |

AngularJS directives

|  |  |
| --- | --- |
| Directives | Description |
| ng-switch | Conditionally instantiate one template from a set of choices, depending on the value of a selection expression. |
| ng-view | The base directive responsible for handling routes that resolve JSON before rendering templates driven by specified controllers. |
| ng-if | Basic if statement directive that allow to show the following element if the conditions are true. When the condition is false, the element is removed from the DOM. When true, a clone of the compiled element is re-inserted. |
| ng-aria | A module for accessibility support of common ARIA attributes. |
| ng-animate | A module provides support for JavaScript, CSS3 transition and CSS3 keyframe animation hooks within existing core and custom directives. |
| ng-table | Simple table with sorting and filtering on AngularJS |

Since ng-\* attributes are not valid in HTML specifications, data-ng-\* can also be used as a prefix. For example, both ng-app and data-ng-app are valid in AngularJS

### 2.3.11 ngResorce.

A factory which creates a resource object that lets you interact with RESTful server-side data sources.

The returned resource object has action methods which provide high-level behaviors without the need to interact with the low level $http service.

Requires the ngResource module to be installed.

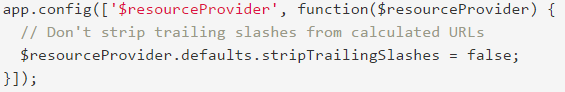
By default, trailing slashes will be stripped from the calculated URLs, which can pose problems with server backends that do not expect that behavior. This can be disabled by configuring the $resourceProvider (Fig.2.1.).

Fig.2.1. $resourceProvider configuration

### 2.3.12 Application server.

An application server is a software framework that provides both facilities to create web applications and a server environment to run them.

Application Server Frameworks contain a comprehensive service layer model. An application server acts as a set of components accessible to the software developer through an API defined by the platform itself. For Web applications, these components are usually performed in the same running environment as their web server(s), and their main job is to support the construction of dynamic pages. However, many application servers target much more than just Web page generation: they implement services like clustering, fail-over, and load-balancing, so developers can focus on implementing the business logic.

In the case of Java application servers, the server behaves like an extended virtual machine for running applications, transparently handling connections to the database on one side, and, often, connections to the Web client on the other.

The role of application server is WildFly.

WildFly, formerly known as JBoss AS, or simply JBoss, is an application server authored by JBoss, now developed by Red Hat. WildFly is written in Java, and implements the Java Platform, Enterprise Edition (Java EE) specification. It runs on multiple platforms.

WildFly is free and open-source software, subject to the requirements of the GNU Lesser General Public License (LGPL), version 2.1.

On 20 November 2014, JBoss Application Server was renamed WildFly. The JBoss Community and other Red Hat JBoss products like JBoss Enterprise Application Platform were not renamed.

## 2.4 Database model.

Entity-relationship model can be used as a basis for unification of different representations of data through the network model, relational model and set of entities.

In the entity-relationship model uses a more natural representation, according to which the real world consists of entities and relationships. This model is based on some important semantic information about the real world (description of other results related to semantics databases. Entity-relationship model can be used as a basis for a unified data presentation.

Conceptual ER-model provides more information on the composition of the database reveals the content database entities and specific fields for each database table, with a clear indication that the field will be the primary key (PK), which is a foreign key (FK) and which fields are required for entry.

Following tables will represent entities that will be used in system database. Following entities will be available for both sub-systems: main application and user portal. The one key difference is that main application will have right to create, edit, delete and receive records, when from other side user portal will have rights only to receive data records from database.

Table 2.3.

Entity Users

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column name | Type | Length | Limitations | Limitations integrity columns |
| User\_ID | INT |  | Unique ID | Primary Key |
| Name | Varchar | 100 | Unique user name | Not null, |
| Role | Varchar | 15 |  | Not null |
| Login | Varchar | 100 |  | Not null |
| Password | Varchar | 100 |  | Not null |

Table 2.4.

Entity Records

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column name | Type | Length | Limitations | Limitations integrity columns |
| Record\_ID | INT |  | Unique ID | Primary Key |
| User\_ID | INT |  |  | Not null, Foreign Key |
| Date | Datetime |  |  | Not null |
| Plain\_ID | INT |  |  | Not null, Foreign Key |
| Sim\_data | Text |  | Simulator big text data type |  |

Table 2.5.

Entity Plain\_Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column name | Type | Length | Limitations | Limitations integrity columns |
| Plain\_ID | INT |  | Unique ID | Primary Key |
| Name | Varchar | 30 | Unique name |  |

Table 2.6.

Entity Metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column name | Type | Length | Limitations | Limitations integrity columns |
| Metric\_ID | INT |  | Unique ID | Primary Key |
| Name | Varchar | 20 |  | Not null |
| Value | Double |  |  | Not null |
| Situation\_ID | INT |  |  | Not null, Foreign Key |

Table 2.7.

Entity Situations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column name | Type | Length | Limitations | Limitations integrity columns |
| Situation\_ID | INT |  | Unique ID | Primary Key |
| Name | Varchar | 30 |  | Not null |
| Description | Varchar | 500 |  |  |

## 2.5 Use-case diagrams representation in main application.

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

While a use case itself might drill into a lot of detail about every possibility, a use-case diagram can help provide a higher-level view of the system. It has been said before that "Use case diagrams are the blueprints for your system".[1] They provide the simplified and graphical representation of what the system must actually do.

### 2.5.1 Instructor use-case diagram.



Fig.2.3.Instructor use-case diagram.

### 2.5.2 Administrator use-case diagram.

Fig.2.4. Administrator use-case diagram.

## Conclusion

Development of web applications is a complicated process involving development of architecture that will be effective in peculiar way, data access layer, which will have to be efficient in process of data transition and data management, and user interface that must be intuitive, logical and comfort for final user.

Commonly used architecture that is time-tested and tested with huge amount of users and companies, shows the efficiency and reliability of client-server architecture.

Usage of MVC pattern isn’t less used, so the popularity shows us how good it can be, how understandable it can be for developed and easy for realization with usage of popular frameworks and technologies. It can be less costly for development process because of amount of best practices and amount of experience developer commonly having in usage of these technologies.

Development of database is easy part but it is very relative to purpose needs. For current peculiar way we need few tables and small amount of relations. But for security needs it is necessary to have two types of connection for different sub-systems. Configuration of local network is as necessary as other parts. Local deployment and local access to main application is part of preventions of security fail or hack.

Dividing into sub-systems is related with security and decries cost of development. Creation of two different limited functionality is more effective than development one wide functional system with increased level of security and extend user interface.

The easiness of development such system is that most of developers are familiar with current technologies and if not, it is possible to learn them in short amount of time and investigate all scopes of development, from architecture to user interface.