1DV404 Iterativ Mjukvaruutveckling.
Laboration 3.
Student: Andréas Anemyr WP -14 (aa223ig).

Tidslog

08:00-18:00	(15/12 -14)	Förstudier/test jsTestDriver	Planerat genomförandet av laboration 3 1DV404
11:30-13:00	(16/12 -14)	Testplan	Satt samman mall för testprojekt. Avses att vidare förbättras i iterat 2!
13:30-17:00	(16/12 -14)	Design & impl	
18:00-23:00	(16/12 -14)	Enhetstestning	
08:00-13:00	(17/12 -14)	Impl. Testsviten och kör	
08:00-11:00	(18/12 -14)	Integrationstestning	
13:00-14:00	(18/12 -14)	Reflektion	

Utöver denna log har det tillkomiit minst! 20 timmar förstudier och diver test av jsTestDriver och tittande på youtubeklipp som handlat om testning på olika sätt. Bla tdd.

Uppgift 1 – Planera

Version 1.0

	Kalkyltid	Verklig tid
Uppgift 1 - Planera	00:30 H:m	00:30 H:m
Uppgift 2 - Testplan	01:30 H:m	01:30 H:m
Uppgift 3 - Design & implementation	02:00 H:m	03:30 H:m
Uppgift 4 - Enhetstestning	02:00 H:m	05:00 H:m
Uppgift 5 - Implementera testsviten och kör	02:00 H:m	05:00 H:m
Uppgift 6 - Integrationstestning	02:00 H:m	03:00 H:m
Uppgift 7 - Refelektion	00:30 H:m	01:00 H:m
Övrigt påläsning för laboration 3 (tester eclipse jsTestDriver mm...)	12:00 H:m	20:00 H:m

Diff kalkyltid – Verklig tid = +27,5 TIM

Sammanfattning och summering ang. planering:

Planeringen gjordes medvetet optimistikt för att sätta ett mål som kunde tänkas var väl högt satt. Sammanfattningsvis kan man dock säga att laborationen tagit tre gånger så lång tid än vad min planering önskade. Återigen finns svart på vitt att planera något som innebär att ta till sig ny kunskap med dess tankebanor är oerhört svårplanerat. Att sedan förstå instruktioner och att jobba mot rätt mål är inte heller självklart på något sätt.

Uppgift 2 – Testplan (testprojekt) - Gymnastiktävlingssystem

Version 1.0

- **Introduktion**
  - Testplanen avser att vara en viktig och primär del i utvecklingen av systemet. En viktig aspekt, samtliga involverade, bör ha i åtanke är att man ska tagit del av och läst testplanen ingående och veta hur den fungerar. Testplanen är en viktig del i kommunikationen mot kunden för att utvecklingen av programvaran ska bli enligt kundens beställning. Sammanfattningsvis kan man säga att detta dokuments testplan är ett primärt verktyget för att uppå ett önskat resultat vid och inför varje kommande release av programvara.
  -
- **Test objekt (att) säkerställa/eftersträva**
  - Kund
    - - Att kundens krav på utvecklad programvara är förankrad hos kunden.
      Detta ska göras regelbundet. Syftet är att minska risken för ”fel” utvecklad programvara
  - Kod
    - - Att kod produceras enligt gemensam standard.
      Syftet är att en gemensam standard avsevärt förenklar vid framtida rekryteringar, felsökningar etc.
    -
- **Egenskaper i testplanen**
  - Use Case
    - implementeras enligt prioriterad lista. Samtliga användningsfall som implementeras ska testas för/under alt. efter implementation.
- **Egenskaper utanför testplanen**
  - Kod
    - Inga tester kommer primärt utföras på prestanda och säkerhet.
- **Tillvägagångssätt**
  - Testplanen är gemensamt framtagen. Dock är att beakta att den är under ständig förändring. Aktuell testplan finns tillgänglig under företagets privata repositore på github.com
  - Kod kommer att enhetstestas med assertions. (jsTestDriver)
  - Eclipse är den primära IDE:n för projektet i sin helhet. Innan resp utvecklare leverar kod för test av delresultat måste den innna testköras mot jsTestDriver.
- **Misslyckat test**
  - Samtliga involverade i utvecklingsteamet är ansvariga för att rapportera händelser/faktorer som kan anses tillföra en ökad risk för projektet i detalj eller som helhet. Testplanen omfattar såväl kod som kommunikation. Samtliga objekt i testplanen anses vara primära och av viktig art. Avvikelse rapporteras enligt standard framtagen i företaget alt. E-posta till utvecklarna@gymnastikligan.se
- **Test delresultat**
  - ^ diskuteras kommande iteration del av IEEE 829 test plan struktur
  - Samtliga delresultat bör lagras persistent som ett resursbibliotek för eventuell framtida behov. Delresultat bör även versionhanteras, framförallt när det gäller kod.
- Att testa
  - ^ diskuteras kommande iteration del av IEEE 829 test plan struktur
- Behov
  - ^ diskuteras kommande iteration del av IEEE 829 test plan struktur
- Responsibilities
  - ^ diskuteras kommande iteration del av IEEE 829 test plan struktur

- Schema
  - Testplanen är dynamisk och under ständig förändring. Under projektets gång kan delar komma att förändras. Delar kan läggas till alternativt plockas bort. Var 14:e dag (måndag morgon) avser utvecklingsteamet genom ett gemensamt möte säkerställa att utvecklingen når gemensamt mål. Under ”utvecklingsmötet” finns testplanen med som en fast punkt.
- Kända och okända risker
  - Ekonomi (Känd risk))
    - Ligan har tidigare haft ekonomiska bekymmer. För att minska risken finns ett avtal om löpande fakturering vartefter utvecklingen framskrider.
  - TDD
    - Eventuell avsaknad av erfarenhet. Projektet ses som en del i utvecklingen att kompetenshöja utvecklarna och teamet i sin helhet.

  - ? (Okända risker)
    - ^ diskuteras kommande iteration del av IEEE 829 test plan struktur. Viktigt!

- Godkännande
  - Eftersom ligan har godkänt en betalningsmodell som innebär fakturering vid varje delrelease, finns därtill en överenskommelse, om att kunden ska godkänna varje release som accepterad. Inga fakturor får skickas till kund innan acceptans av kund givits till fullo. Finns inget godkännande på sista delresultat efter redovisning/delresultat kommer utvecklingen att pausas intill acceptans givits.

(Inspirationskälla testplan (http://en.wikipedia.org/wiki/Test_plan, indelning enligt IEEE 829 test plan struktur)

Uppgift 3 – Design och implementation

Version 1.0

Reducering kommer med stor sannolikhet att genomföras på användningsfallen avseende lab 3. Det har kommit till sin klarhet vid närmare anblick att användningsfallen är omfattande. De kommer att behöva brytas ned till en mycket lägre nivåer för att förenkla implementation av klasser och därmed även senare underlätta arbete med att omvandla användningsfallen till testfall i ett senare skede. Första användningsfallet (Loggar in ^ registrerar klubbadministratör) kommer att brytas ned till att omfatta istället två mindre användningsfall. Därtill kommer primärt och alternativt flöde att förkortas och egentligen innebära att två helt nya användningfall inom det ursprungliga, dock med samma aktör (Systemadministratör). De nya användningfallen kommer att benämnas ”Loggar in” & ”Registrera klubbadministratör”. Eftersom användningsfall är förankrade med kund kommer inte nya diagram brytas ned. De ska ses som ett recept som ska följas. Nedbrytningen i sig är synliggjord genom att punkt 1 får underrubrik såsom 1 (1) & 1 (2). På detta sätt blir angreppsvägen tydligare. Allt för att uppnå de krav som framkommit i visiondokumentets.

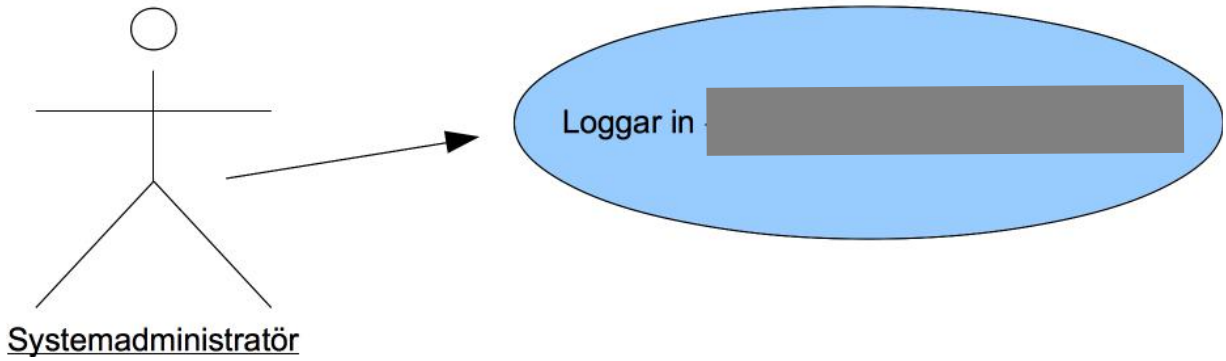
punkt	Aktör	Användningsfall
1(1)	Systemadministratör	Loggar in
1(2)	Systemadministratör	Registrera klubbadministratör

Användningsfall ^ punkt 1(1) ^ Design & Implementation use case: Loggar in

Namn	Beskrivning	Uppgifter
Systemadministratör	En person som utsetts av ligan	Ansvarig för att introducera nya klubbar genom elektronisk inbjudan. Även allmän bevakning av systemet

Beskrivning:

Systemadministratören loggar in i systemt genom att ange användarnamn och lösenord



Berättelse/Scenario

Per = Systemadministratör.

För att erhålla en hög säkerhet i systemet måste man logga in i systemet för att kunna bruka systemt. Som systemadministratör finns det en mängd med rutiner som måste vara avskilt för systemets övriga brukare.

Pre
Systemadministratör registrerad i systemet.

Post
Systemadministratören är inloggad i systemet.

Primär

1. Systemadministratör matar in användarnamn
2. Systemadministratören matar in lösenord
3. Systemet kontrollerar inmatat data mot databas med användare
4. Systemanvändaren släpps in i systemet

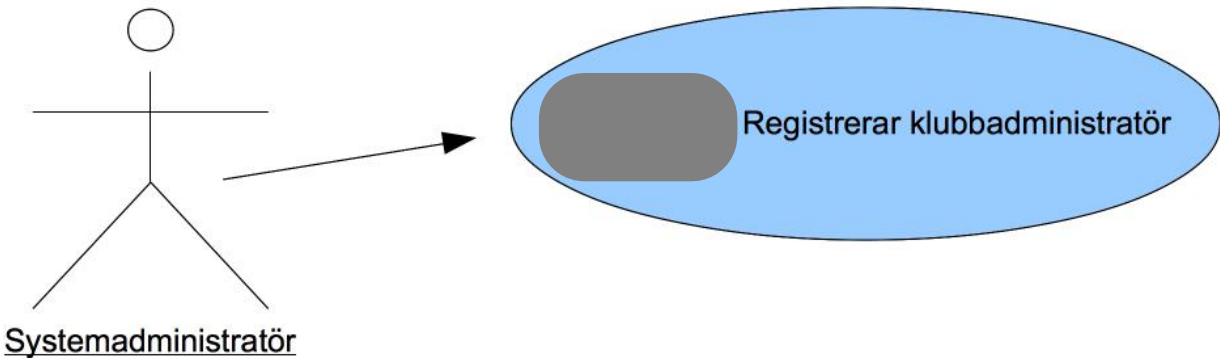
Alternativt flöde:

1. Enligt primärt flöde
2. Enligt primärt flöde
3. Enligt primärt flöde
4. Fel användarnamn alt. lösen
5. Systemadministratör erbjuds ny inloggning

Användningsfall ^ punkt 1(2) ^ Design & Implementation use case: Registrerar klubbadministratör

Namn	Beskrivning	Uppgifter
Systemadministratör	En person som utsetts av ligan	Ansvarig för att introducera nya klubbar genom elektronisk inbjudan. Även allmän bevakning av systemet
Klubbadministratör	En person som utsetts av klubben	Ansvarig för klubbens allmäna data. Den person som primärt håller dialog med systemadministratören. Även befogenhet att delegera bort rollen lagadministratör

Beskrivning:
Systemadministratören lägger till e-postadress i databasen



Berättelse/Scenario

Per = Systemadministratör.
Eftersom systemet i sig självt har tillgång til metoder som automatiskt gör det möjligt att bjuda in nya klubbadministratörer behöver Per logga in i systemet för att lägga till en potentiell blivande klubbadministratör. För att kunna lägga till detta behöver han logga in och därtill ha tillgång till
E-postadressen för den blivande klubbadministratören.

Pre
Systemadministratör inloggad i systemet.
Systemadministratören har tillgång til e-postadress

Post
Systemet har tillgång till en ny e-postadress i syte att bjuda in en klubbadministratör.

Primär

1. Systemet ansluter till databas om inte anslutning finns
2. Systemadministratör triggar event om att lägga till ny klubbadministratör
3. Systemt lägger till ny användare med unikt ID.

Alternativt flöde:

1. Systemet misslyckas med anslutning
2. Generera fellogg

Konstruktorfunktion Database

Källfil: src/Database.js

Kommentar:
Implementation av ”klassen” Database är i grunden en konstruktorfunktion i javascript. Även om js i grunden inte har stöd för klassen och metoder på samma sätt som exempelvis c# har det ändå funnits en strävan åt att likna den ”klassiska” programmeringen inom oop.

users: simulerar en databas i form av en array. Tanken är att applikationen byggs iterativt och databas kopplas på i ett senare skede.

getUsers(): Metod som har som uppgift att returnera samtliga fält databasen från ”tabellen” users

addEmail(): Metod som används för att registrera in en ny ”user” i databasen. Metoden genererar automatisk ett inkrementerat id med utgång från hur många users det för närvarande finns i systemet.

Database
users : array
getUsers() : array addEmail()

```
”use strict";
//Will be replaced by a real dbClass. Just simulating!
function Database(){

	this.users = [
	{
		id:			1,
		fname:		"Erik",
		lname:		"Eriksson",
		passWord:	2222,
		eMail:		"erik@eriksson.se",
		isAdmin:	true,
	},
	{
		id:			2,
		fname:		"Anders",
		lname:		"Andersson",
		passWord:	1111,
		eMail:		"erik@eriksson.se",
		isAdmin:	false,
	}
	];

};

Database.prototype.getUsers = function(){
	return this.users;
};

Database.prototype.addEmail = function(eMail){
	this.users.push({id: this.users.length + 1, eMail: eMail});
};
```

Konstruktorfunktion SportClub

Källfil: src/SportClub.js

Kommentar:
Implementationen av klassen (konstruktorfunktionen) SportClub får i detta begynnande skede vara så minimal den kan vara. Finns ingen orsak att låta den svämma över av en massa funktionalitet som ”eventuellt” kommer att behövas så småningom. Att hålla kodabasen men samtidigt uppstå kraven för projektet borde göra underhåll av koden enklare. Anledningen till att jag måste ha med den här är att användningfall 1 i visionsdokumentet kommer att vara beroende av att det går att registrerat sig för ett lag om man ska kunna registrera sig som en klubbadministratör. Inledningsvis är det tillräckligt om det finns fält för lagets namn och ett unikt lag id. Tanken är att Systemet frågar databasen efter ett eller flera lag därefter instansieras objekt utiftrån databassvaret som givits. Instansiering sker genom anrop till konstruktorfunktionen SportClub som återfinns i SportClub.js

SportClub
clubName : string clubAdminId
getClubAdminId() : number setClubAdminId() : void getName() : string setName() : void

```
function SportClub(clubName, clubAdminId){

	this.setClubAdminId = function(_clubAdminId){
		clubAdminId = _clubAdminId;
	};

	this.getClubAdminId = functin(){
		return clubAdminId;
	};

	this.setName = function(_clubName){
		clubName = _clubName;
	};

	this.getName = function(){
		return name;
	};
};
```

Uppgift 4 – Enhetstestning

Kommentar gällande uppgift 4:
Att göra en komplett tabell på alla möjliga scenarior är svårt. Det är svårt att få in all tänkbar och icke tänkbar logik. Försök har gjorts att uppnå en specifikation av enhetstestning som ställer kommande krav vid implementation av tester.

Alternativet till denna metod kunde varit en annan tabelluppbyggnad men känslan är att jag får in mycket information på liten yta men att den trots allt är greppbar.

Motivering Enhetstestning 4:
Försök har gjort att uppnå en enhetstestning som innebär minskade risker vid utveckling av systemet. Beaktning har även tagits till övriga och förhoppningsvis samtliga krav som kunden förutsätts kräva vid leverans. En bra utgångspunkt är att försöka skriva kod på ett sätt som stärker oddsen för en framtida acceptans vid leverans.

Uppgift 4 Enhetstestning

Enhetstestning avseende **Login:**

Kommentar:
Funktionen login ligger inte inledningsvis i utvecklingen implementerad i en egen eller annan klass. Funktionen är deklarerad som medlem i objektet init som laddas när webbläsaren anropar eventet ”load”. Eventet är satt med addEventListener().

Anledning till att den ligger med så här tidigt är att all funktionalitet kräver att man är inloggad. Tanken är att funktionen får sina inloggningsparametrar genom en manuell inmatning från brukare av systemet när det är driftat. Dock är systemet inledningsvis försett med en hårdkodad testanvändare (test@test.se) i klassen database för att funktionalitet ska kunna testas.

Testfixtur: Login
Testsvit: testLogin (jsTestDriver)
Version: 1.0

Anrop	Förväntat	Kommentar	testID	Check
init.loginUser(”Erik”, 1111)	fail	Fel lösen	1.1	X
init.login(”Erik”, 2222)	pass	Anv. I databas	1.2	X
init.login(””)	fail		1.3	X
Init.login(”nisse”, 2222)	fail	Anv. Existerar inte!	1.4)	X

Enhetstestning avseende **Database:**

Kommentar:
För att kunna utför enhetstestning på klassen database genereras först en referansvariabel benämnd db nedan.

Testfixtur: Database
Testsvit: testDatabase (jsTestDriver)
Version: 2.0

Anrop	Förväntat	Kommentar	testId	Check
db === new Database()	pass	Vid db === null ska ett assert genereras!	2.1	X
db.addEmail(”test@test.se”)	pass	AddEmil lägger till framtida klubbadministratör med unikt uppräknatd baserat på antal user i database	2.2	X
db.addEmail(1234)	fail	Assert nyttjar Tools	2.3	X
db.addEmail(1234); db.addEmail(test@test.se)	pass	Testet ska simulera att göra två försök. Varvid det andra ska godkännas som valid data.	2.4	X

Enhetstestning avseende **Tools:**

Kommentar:
Klassen Tools avser att samla upp metoder som systemet behöver nyttja på en mängd olika ställen i programkod.

Testfixtur: Tools
Testsvit: testTools (jsTestDriver)
Version: 3.0

Anrop	Förvänt at	Kommentar	testId	Check
tools === new Tools()	pass	Vid tools === null ska ett assert genereras!	3.1	X
tools.validateEmail(”test@test.se”)	pass		3.2	X
tools.validateEmail(243564)	fail		3.3	X

Uppgift 5 Implementera testsviten och kör

Testfixtur:	**Login**
Testsvit:	**testLogin (jsTestDriver)**
Version:	**1.0**

Allmän kommentar testmiljö:
Eclipse används som IDE. För att implementera testsviter och testkod används ett plugin till eclipse som heter jsTestDriver utvecklat av google. Det går alltså inte att köra denna kod direkt i en js fil med en förväntan om att det ska fungera. Pluginmodulen möjliggör att testa sin js mot samtliga browser genom att endast spara filen (går att koppla bort denna funktionen.

```
/*      ************************************
 *      Testfixtur:    Login
 *      TestSuite:     testLogin
 *      Version:       1.0
 *  ************************************/

        testLogin = TestCase("testLogin");

//testID 1.1
testLogin.prototype.testLoginId1dot1 = function(){
        db = new Database();
        init.users = db.getUsers();
        init.loginUser("Erik", 1111);
        assertNull("testID 1.1: Wrong Password. Check init.loginUser. User should not be logged
in!", init.currentUser);
        //Reset variables to init state befor next test in this suite
        db = null;
        init.currentUser = null;
};
//testID 1.2
testLogin.prototype.testLoginId1dot2 = function(){
        db = new Database();
        init.users = db.getUsers();
        init.loginUser("Erik", 2222);
        assertSame("testID 1.2: User should be logged in. Test tests user: Erik & password: 2222.
It's listed inte database!", "Erik", init.currentUser.fname);
        db = null;
        init.currentUser = null;
};
//testID 1.3
testLogin.prototype.testLoginId1dot3 = function(){
        db = new Database();
        init.users = db.getUsers();
        init.loginUser("");
        assertNull("testID 1.3: Becaus params not is filled in when calling method init.currentUser
should be Null!", init.currentUser);
        db = null;
        init.currentUser = null;

};
//testID 1.4
testLogin.prototype.testLoginId1dot4 = function(){
        db = new Database();
        init.users = db.getUsers();
        init.loginUser("nisse", 2222);
        assertNull("testID 1.4 User should not be logged in is not valid!!", init.currentUser);
        db = null;
        init.currentUser = null;
};
```

Testfixtur: **Database**
Testsvit: **testDatabase(jsTestDriver)**
Version: **2.0**

Allmän kommentar testmiljö:
Eclipse används som IDE. För att implementera testsviter och testkod används ett plugin till eclipse som heter jsTestDriver utvecklat av google. Det går alltså inte att köra denna kod direkt i en js fil med en förvänta om att det ska fungera. Pluginmodulen möjliggör att testa sin js mot samtliga browser genom att endast spara filen (går att koppla bort denna funktionen.

```
"use struct";
/*      ************************************
 *      Testfixtur:    Database
 *      TestSuite:     testDatabase
 *      Version:       2.0
 *  ************************************/

        testDatabase = TestCase("testDatabase");

//testID 2.1
testDatabase.prototype.testDatabaseId2dot1 = function(){
        var db = new Database();
        assertTrue("Failed initalize new object of Database", Boolean(db));
        db = null;
};
//testID 2.2
testDatabase.prototype.testLoginId2dot2 = function(){
        var db = new Database();
        db.addEmail("test@test.se");
        assertEquals("testID: 2.2 A valid \"test@test.se\" failed to add but is valid",
                                "test@test.se", db.users[db.users.length -1].eMail);
        db = null;
};
//testID 2.3
testDatabase.prototype.testLoginId2dot3 = function(){
        var db = new Database();
        assertFalse("testID 2.3: Fail! 1234 is not a valid eMail.", db.addEmail(1234));
        db = null;
};
//testID 2.4
testDatabae.prototype.testLoginId2dot4 = function(){
        var db = new Database();
        assertFalse("testId 2.4.1: Failed because an invalid adress was added", db.addEmail(1234));
        assertTrue("testId 2.4.2: Failed to add eMail \"test@test.se\"",
db.addEmail("test@test.se"));
        db = null;
};
```

Testfixtur: **Tools**
Testsvit: **testTools(jsTestDriver)**
Version: **3.0**

Allmän kommentar testmiljö:
Eclipse används som IDE. För att implementera testsviter och testkod används ett plugin till eclipse som heter jsTestDriver utvecklat av google. Det går alltså inte att köra denna kod direkt i en js fil med en förvänta om att det ska fungera. Pluginmodulen möjliggör att testa sin js mot samtliga browser genom att endast spara filen (går att koppla bort denna funktionen.

```
"use strict";
/*      ************************************
 *      Testfixtur:    Tools
 *      TestSuite:     testTools
 *      Version:       3.0
 *  ************************************/

        testTools = TestCase("testTools");

//testID 3.1
testTools.prototype.testToolsId3dot1 = function(){
        var tools = new Tools();
        jstestdriver.console.log(typeof tools);
        if(!tools){
                fail("testId 3.1 Failed to initalize object from Tools");
        }
};
//testID 3.2
testTools.prototype.testToolsId3dot2 = function(){
        var tools = new Tools();
        assertTrue("testId 3.2: ",tools.validateEmail("test@test.se"));
};
//testID 3.3
testTools.prototype.testToolsId3dot3 = function(){
        var tools = new Tools();
        assertFalse("testId 3.3: ",tools.validateEmail(243564));
};
```

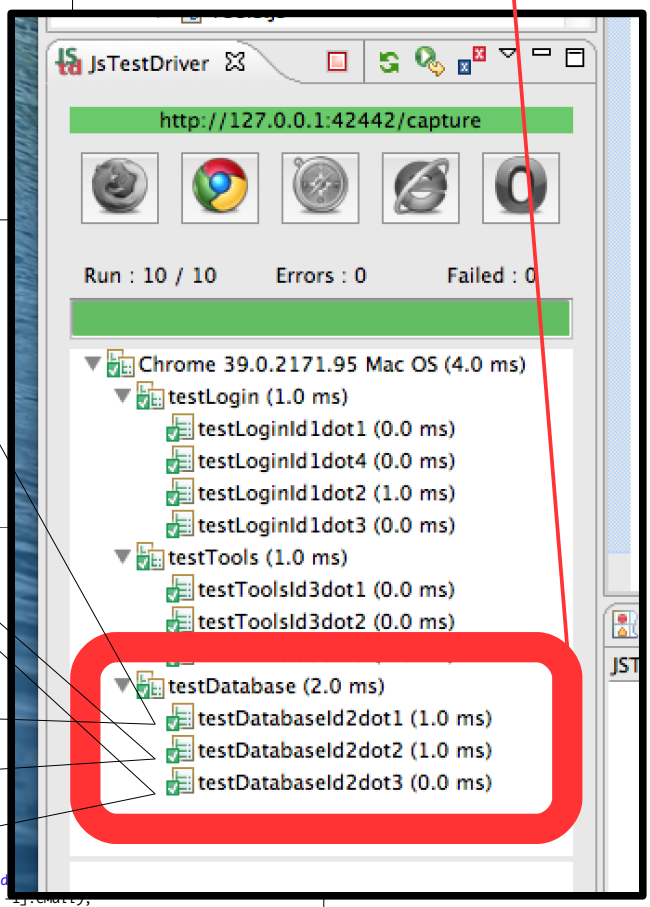
Uppgift 6 Integrationstestning – Avseende Database

```
"use strict";
//Will be replaced by a real dbClass. Just simulating!
function Database(){

	this.SportClubs = [
	{
	    name:			"testLaget",
	    clubAdminId:	1,
	}
	];

	this.users = [
	{
		id:				1,
		fname:			"Erik",
		lname:			"Eriksson",
		passWord:	2222,
		eMail:			"erik@eriksson.se",
		isAdmin:	true,
	},
	{
		id:				2,
		fname:			"Anders",
		lname:			"Andersson",
		passWord:	1111,
		eMail:		"erik@eriksson.se",
		isAdmin:	false,
	}
	];

};

Database.prototype.getUsers = function(){
	return this.users;
};

Database.prototype.addEmail = function(eMail){
	var tools = new Tools();
	if(tools.validateEmail(eMail))
	{
		this.users.push({id: this.users.length + 1, eMail: eMail});
		return true;
	}
	else
	{
		return false;
	}
};
```

Förklaring:
Följer man pilarna kan man tydligt se kopplingen mellan testfall och vilken klass det är som är involerat i testet.

Not.
Varje test har ett unikt testid som återkopplar till testfall uppgift 4.

Övrigt
JsTestDriver har marketerat att att samtliga test gått igenom med förväntat resultat och samtliga assertions får anses att ha gått igenom. (extra kontrolerl har även genomförs genom att skicka felaktiga parametera på resp. assert och då markeras detta som fail).




```
"use strict";
/*		**************************************
 *		Testfixtur: Database
 *		TestSuite:  testDatabase
 *		Version:    2.0
 *  **************************************/

	testDatabase = TestCase("testDatabase");

//testID 2.1
testDatabase.prototype.testDatabaseId2dot1 = function(){
	var db = new Database();
	assertTrue("Failed initalize new object of Database", Boolean(db));
	db = null;
};
//testID 2.2
testDatabase.prototype.testDatabaseId2dot2 = function(){
	var db = new Database();
	db.addEmail("test@test.se");
	assertEquals("testID: 2.2 A valid \"test@test.se\" failed to add but is valid
				"test@test.se", db.users[db.users.length - 1].eMail);

	db = null;
};
//testID 2.3
testDatabase.prototype.testDatabaseId2dot3 = function(){
	var db = new Database();
	assertFalse("testID 2.3: Fail! 1234 is not a valid eMail.", db.addEmail(1234));
	db = null;
};
//testID 2.4
testDatabae.prototype.testDatabaseId2dot4 = function(){
	var db = new Database();
	assertFalse("testId 2.4.1: Failed because an invalid adress was added", db.addEmail(1234));
	assertTrue("testId 2.4.2: Failed to add eMail \"test@test.se\"", db.addEmail("test@test.se"));
	db = null;
```

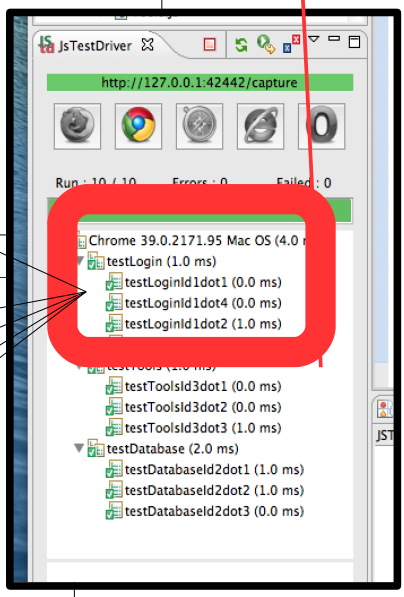
Uppgift 6 Integrationstestning – Avseende Login (init)

```
"use strict";
var init = {
                        currentUser: null,
                        dbConnect: null,
                        tools: null,
                        users: null,

                        //*****************************************************
                        run: function(userName, passWord){
                        this.addEvents();
                        this.tools = new Tools();
                        this.dbConnect = new Database();

                        this.users = this.dbConnect.getUsers();

                        this.loginUser(userName, passWord);

                        },
                        //*****************************************************

                        addEvents: function(){
                                    addEventListener("click", this.newClubAdmin,
                        },

                        newClubAdmin: function(){
                                    var email = prompt("Ange e-postadress");
                                    if(init.tools.validateEmail(email)){
                                                init.dbConnect.addEmail(email);
                                    }
                                    else{
                                                alert("Ange en giltig e-postadres
                                                init.newClubAdmin();
                                    };
                        },
                        loginUser: function(userName, passWord){
                                    this.users.forEach(function(user){
                                                if(user.fname === userName && user.passWord === +passWord)
                                                {
                                                            init.currentUser = user;
                                                }
                                    });
                        }
};

window.onload = function(){
            while(init.currentUser === null){
                        //init.run(prompt("User"), prom    Pass"), true);
                        init.run("Erik", 2222);
            }
};
```

Förklaring:
Följer man pilarna kan man tydligt se kopplingen mellan testfall och vilken klass det är som är involerat i testet.

Not.
Varje test har ett unikt testid som återkopplar till testfall uppgift 4.

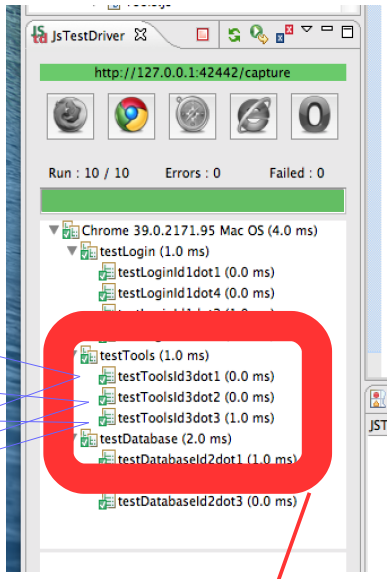
Övrigt
JsTestDriver har marketerat att att samtliga test gått igenom med förväntat resultat och samtliga assertions får anses att ha gått igenom. (extra kontrolerl har även genomförs genom att skicka felaktiga parametera på resp. assert och då markeras detta som fail).




```
"use strict";
/*          ************************************
 *          Testfixtur: Login
 *          TestSuite:  testLogin
 *          Version:    1.0
 *  ************************************/

            testLogin = TestCase("testLogin");

//testID 1.1
testLogin.prototype.testLoginId1dot1 = function(){
            var db = new Database();
            init.users = db.getUsers();
            init.loginUser("Erik", 1111);
            assertNull("testID 1.1: Wrong Password. Check init.loginUser. User should not be logged in!",
init.currentUser);
            //Reset variables to init state befor next test in this suite
            db = null;
            init.currentUser = null;
};
//testID 1.2
testLogin.prototype.testLoginId1dot2 = function(){
            var db = new Database();
            init.users = db.getUsers();
            init.loginUser("Erik", 2222);
            assertSame("testID 1.2: User should be logged in. Test tests user: Erik & password: 2222. It's listed inte
database!", "Erik", init.currentUser.fname);
            db = null;
            init.currentUser = null;
};
//testID 1.3
testLogin.prototype.testLoginId1dot3 = function(){
            var db = new Database();
            init.users = db.getUsers();
            init.loginUser("");
            assertNull("testID 1.3: Becaus params not is filled in when calling method init.currentUser should be
Null!", init.currentUser);
            db = null;
            init.currentUser = null;

};
//testID 1.4
testLogin.prototype.testLoginId1dot4 = function(){
            db = new Database();
            init.users = db.getUsers();
            init.loginUser("nisse", 2222);
            assertNull("testID 1.4 User should not be logged in is not valid!!", init.currentUser);
            db = null;
            init.currentUser = null;
```

Uppgift 6 Integrationstestning – Avseende Tools

```
"use strict";
/*          ***************************************
 *          Testfixtur: Tools
 *          TestSuite:  testTools
 *          Version:    3.0
 *  ***************************************/

            testTools = TestCase("testTools");

//testID 3.1
testTools.prototype.testToolsId3dot1 = function(){
            var tools = new Tools();
            jstestdriver.console.log(typeof tools);
            if(!tools){
                        fail("testId 3.1 Failed to initalize object from Tools");
            }
};
//testID 3.2
testTools.prototype.testToolsId3dot2 = function(){
            var tools = new Tools();
            assertTrue("testId 3.2: ",tools.validateEmail("test@test.se"));
};
//testID 3.3
testTools.prototype.testToolsId3dot3 = function(){
            var tools = new Tools();
            assertFalse("testId 3.3: ",tools.validateEmail(243564));
};
```




```
function Tools(){
            //Empty block!
}

Tools.prototype.validateEmail = function(email) {
    var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
    return re.test(email);
};
```

Förklaring:
Följer man pilarna kan man tydligt se kopplingen mellan testfall och vilken klass det är som är involerat i testet.

Not.
Varje test har ett unikt testid som återkopplar till testfall uppgift 4.

Övrigt
JsTestDriver har marketerat att att samtliga test gått igenom med förväntat resultat och samtliga assertions får anses att ha gått igenom. (extra kontrolerl har även genomförs genom att skicka felaktiga parametera på resp. assert och då markeras detta som fail).

Förtydligande/motivering avseende illustarationer (uppg 6):

Eftersom en bild säger mer än tusen ord valde jag att illustrera hur testsviterna hänger ihop genom att illustrera detta med pilar som påvisar sammanhanget mellan testsvit och den klass som avser att testas. I uppgift 4 avsågs man ta fram specifika tester för att sedan implementera i uppgift 6. Till testfallen försökte jag även plocka ut ett antal möjligt scenario som eventuellt skulle kunna inträffa vid körning av kod. Framförallt har jag testat att skicka valida och icke valida argument in till metoder. Det bör dock betonas att jag uteslutande använt mig av tidigare förutbestämda testfallsdata. Eftersom jag anser det redundant att återigen rada upp tabeller för att redovisa det faktiskta resultatet gentemot förväntat eftersom samtliga tester kunnat genomföras utan synliga uppkomna brister/problem anser jag att samtliga testsviter håller god kvalité och om man ändå vill utforska vilka parametrar som skickats kan man alltså leta fram dessa i uppg 3 testfall.

Ska man ändå ta detta till en ”nästa” nivå för att förbättra dokumentering så bör man trots allt även skapa en tabell för redovisning av testfallsdata. Jag anser nog att detta skulle göras effektivas genom att använda sig av de tabeller som skapades i uppgift tre. Men till dessa skulle man kunnna komplettera upp ett antal kolumner som säger vem som utfört test, när test utförts. Eventuella avvikelser mot det förväntade resultatet som då skulle kunna användas som ett underlag för återrapportering till utvecklingsteamet och påtala eventuella bugga alternativt brister.

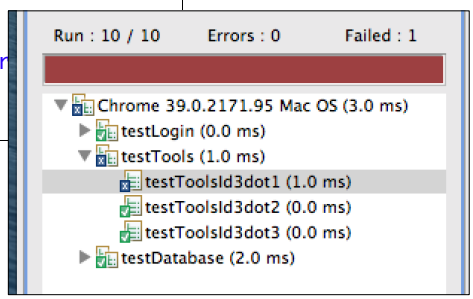
Angående kommentare i teskod så har dessa hållit till minium beroende på att försök har gjorts att använda sig av meddelandet som assertet generera istället. Då svämmar inte koden över av kommentarer som blivit redundant eftersom meddelandet (oftast) trots allt ändå säger vad koden förväntas kontrollera.

Testfall och testdatan utgick ifrån de underlag som tagits fram för uppgift 3. Tankengången var att bearbeta samtliga testfall som tagits och att dessa skulle utföras enligt underlag för test. Målet kan sägas vara väl uppfyllt samtliga testfall kunde genomföras med förväntat resultat utan några undantag alls.

Avvikande assert

Nedanstående skärmdumpar är endast en illustration för hur jsTestDriver beter sig om ett assert avviker mot data som förväntats. Samtliga testfall har simulerats till motsatt kontroll. Exempelvis assertNul har bytts till assertNotNull. Avsikten med detta var att testa om stärka för ytterligarre bevis om att testerna är och var funktionsdugliga när de kodades.

```
//testID 3.1
testTools.prototype.testToolsId3dot1 = function(){
        var tools = new Tools();
        jstestdriver.console.log(typeof tools);
        if(tools){
                fail("testId 3.1 Failed to initalize object fr
        }
};
```



Uppgift 7 – Reflektion

Iterativ Mjukvaruutveckling. Ett stort ämne med alla möjliga lösningar. Lite sån är min känsla över detta omfattande ämne. Deluppgifterna i sig kanske egentligen inte varit speciellt tuffa om det endast skulle utföras just som delmoment med mindre inlämningar.

Däremot anser jag utmaningen totalt sett i denna laboration varit oerhört krävande. Det har varit svårt att tydligt se ett mål för det man faktiskt ska åstadkomma. Jag har garanterat lärt mig både det ena och det andra under laborationens genomföranden. Det svåra tycker jag har varit att avgränsa sig, veta vad man ska fokusera på och det man lämnar åt sidan nu eller för alltid.

För mig har det varit en stor tillgång att titta på youtubeklipp som avser testning av mjukvara och mjukvaruprojekt. Ibland kör man in på ett spår och fastnar tankemässigt och då brukar jag titta på ett eller flera klipp eventuellt kanske slänga iväg en fråga till en kurskamrat för att se hur de avser att lösa uppgiften. Det resulterar ofta i nya krafter och att man ser lösning på det som ska utföras på ett fräschare sätt.

När labben kändes som mest motig var när jag skulle implementera testsviter och testköra. Här fick jag verkligen en aha-känsla och kunde känna att tester i kod absolut är något man skulle arbeta med så oerhört mycket mer. En av de största tillgångarna med en testsvit måste vara att den kontroller kod vid implemetnterings tidpunkt, men även tiden framöver sålänge som testsviten finns aktiverad. Jag valde/hittade en plugin som google utvecklat (jsTestDriver) som kör igenom testsviterna varje gång man sparar dokumentet. Det intressanta med denna modul är att man inte behöver hoppa mellan IDE:n och webbläsarens konsoll. Vid uppstart av modulen ligger en eller flera webbläsare som slav under modulen och testerna kan allstå genomföras på samtliga webbläsare utan att faktiskt ens behöva hoppa över till webbläsaren. Man kan tom testa en webbläsare på en annan dator. Om man vill så är det möjligt att koppla jasmines (testmiljö för javscript) testmiljlö under jsTestDriver vilket jag säkerligen kommer att testa på framvöver.

Slutligen. Utmaning nu är att försöka få till ett strukturerat tänk om hur man bör/ska gå tillväga inför nya projekt vare sig det är inom skolan, privat eller inom jobbet. Känns trots allt som ett nytt och bättre tänk håller på att växa till sig inom mig om hur jag bör gå tillväga inför nya projekt för att de ska bli lyckosamma ett bättre och effektivare sätt...