



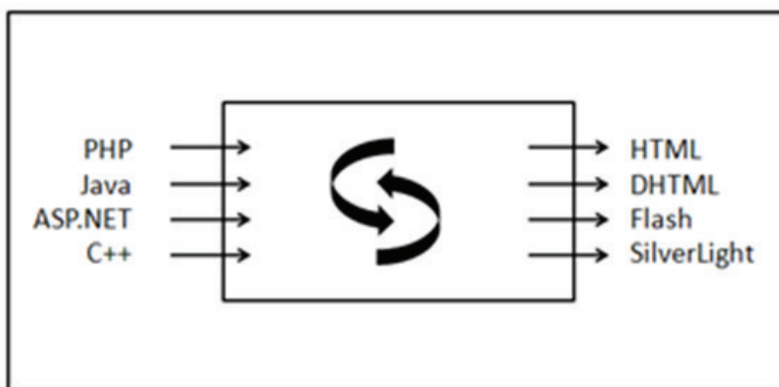
ЕВРОПЕЙСКИ СЪЮЗ

Европейски Социален Фонд 2007 – 2013 Министерство на труда и социалната политика
Оперативна програма „Развитие на човешките ресурси“ BG051PO001 – 7.0.07 „Без граници –
Компонент 1“ – Фаза 2 Договор BG051PO001-7.0.07-0235-C0001 „Създаване на нови умения в
служителите на фирма „Дейтасист“ ООД за прилагане на иновативен подход при предлагане
на онлайн услуги в партньорство с фирма “SC Networx SRL” се финансира от Оперативна
програма „Развитие на човешките ресурси“ чрез Европейски Социален Фонд 2007 – 2013 и
бюджета на Р България.



Европейски социален фонд

WEB БАЗИРАНА МЕТОДОЛОГИЯ ЗА СЛУЖИТЕЛИТЕ НА ДЕЙТАСИСТ ООД



Април, 2014

Бенефициент: «Дейтасист» ООД

Този документ е създаден по проект №BG051PO001-7.0.07-0235 „Създаване на нови умения в служителите на фирма „Дейтасист“ ООД за прилагане на иновативен подход при предлагане на онлайн услуги в партньорство с фирма „SC Networx SRL“. Документът е създаден с финансовата подкрепа на Оперативна програма „Развитие на човешките ресурси“, съфинансирана от Европейския съюз чрез Европейски социален фонд и от националния бюджет на Република България.

Цялата отговорност за съдържанието на документа се носи от «Дейтасист» ООД и при никакви обстоятелства не може да се приема, че този документ отразява официалното становище на Европейския съюз и Договарящия орган.

Разработено от: АДРА ЕООД

гр. Русе, пл.СВЕТА ТРОИЦА, бл. СМК, вх. А, ет. 4, ап. 22, тел.: +359 82 827 145, e-mail:
office@adra-bg.com

АДРА

СЪДЪРЖАНИЕ:

1. Методология. Типизиране на проблематиката при експлоатация на техническо и комуникационно оборудване
2. Използване на прототипизиращи механизми
3. Разрешаване на комуникационни проблеми и такива с компютърните активи на клиента
4. Препоръки за развитие на техническата инфраструктура на клиента при отчетени комуникационни проблеми и такива с компютърните активи на клиента
5. Систематизиране на възможните технологии, използвани от клиента и подробното им описание за приоритизиране на конкретния проблем или възел от дейността
6. Обобщаване на техническите параметри за преминаване към иновативния метод за предлагане на онлайн поддръжка и да детайлизира експертния път, по който служителите на Дейтасист ООД да внедрят новия начин на предлагане на услугата
7. Организация на данните и информационните процеси по предлагане на онлайн поддръжка от начало до край
8. Визуализиране на взаимодействието на всички елементи и процеси при предлагане на услугата

АДРА



МЕТОДОЛОГИЯ. ТИПИЗИРАНЕ НА ПРОБЛЕМАТИКАТА ПРИ ЕКСПЛОАТАЦИЯ НА ТЕХНИЧЕСКО И КОМУНИКАЦИОННО ОБОРУДВАНЕ

Предназначение на Методологията

Целта на настоящата Методология е да се създаде единен, унифициран подход за внедряване на иновативен метод на работа при типизиране на проблематиката при експлоатация на техническо и комуникационно оборудване при предоставяне на услугата индивидуализирана онлайн поддръжка на клиентски мрежи и устройства.

Методологията съдържа логическа последователност от етапи, стъпки, предпоставки, методи и техники, резултати, приложими при внедряване на нови методи на работа в сферата на ИКТ услугите.

Методологията представлява инструмент, който може да бъде прилаган за решаването на различни задачи:

- типизиране на проблематиката при експлоатация на техническо и комуникационно оборудване,
- използване на прототипизиращи механизми, които са в основата на иновативния подход при извършване на дейности по поддръжката на клиенти и по-конкретно при разрешаване на комуникационни проблеми и такива с компютърните активи на клиента,
- консултации, препоръки за развитие на техническата инфраструктура на клиента и др.
- систематизиране на възможните технологии, използвани от клиента и подробното им описание с цел приоритизиране на конкретния проблем или възел от дейността, което автоматично води до превенция срещу технически аварии за критични процеси от дейността на клиента. Тази методология ще очертае
- идентифициране на възможности за усъвършенстване на процесите
- съкращаване на ненужни стъпки, облекчаване на документооборота, преразпределение на отговорностите и др.

Контекст

Дейтасист ООД въвежда иновативен метод на работа, основан на използването на специфична интегрирана технология за генериране на малки софтуерни приложения. Технологията позволява на използващите да ползват технологията или средствата за програмиране, които познават най-добре, дори и те да не са най-приложимите в конкретния случай, като получат резултат от ползването на средствата за програмиране, най-подходящи за определения казус. Целевата технология ще доведе до индивидуализиране на предлаганата услуга онлайн поддръжка на клиентски мрежи и устройства, тъй като обезпечава максимално конкретен подход към клиента, съобразен с наличната му хардуерна и софтуерна технология, специфика на дейността и нужда от онлайн поддръжка. Поради присъщите си характеристики, използването на новата технология ще доведе и до съкращаване на редица разходи, такива като време, разходи за логистика и разходи за наемане на външни консултанти по определени казуси, свързани с непознати технологии, и в резултат на това ще повиши качеството на предлаганата услуга и конкурентоспособността на фирма Дейтасист. Непосредствен ползвател на внедряваната технология ще бъдат служителите на Дейтасист ООД с експертни длъжности с аналитични функции. За постигане на максимален ефект от използването на внедряваната технология решаващо значение има познаването от служителите с експертни длъжности с аналитични функции на принципите, на които е изградена и опознаване на потребителския интерфейс.

Това води до необходимостта от въвеждането и прилагането на единна специфична методология, която да подпомага типизиране на проблематиката при експлоатация на техническо и комуникационно оборудване, систематизиране на възможните технологии, използвани от клиента и подробното им описание.



Обосновка

Пряката, непосредствена връзка на взаимодействие в аспект „необходимост - способност“ между проблематиката, свързана с повишаване качеството на предоставяната услуга и конкурентоспособността на фирмата, от една страна и технологичните възможности, които предоставя целевата технология, от друга страна, дефинира водещия и основен принцип на специфична методология за внедряване на индивидуализирана онлайн поддръжка: пряката, непосредствена права зависимост между достатъчната (пределната) способност за използване предимствата на целевата технология от контактната група и максимизирането на ефективността на производствения процес, респективно на пазарните показатели на фирмата. Така принципът определя и процеса на методологията:

Обучение → Проверка → Допълнително обучение → Проверка....
Натрупване →

В качествен аспект методологията има цикличен, итеративен и инкрементален характер и изглежда по следния начин:

Ниво, умения → Рекапитулация, допълнителна квалификация → По високо ниво, по широка гама умения → Рекапитулация, допълнителна квалификация
→ По високо ниво, по широка гама умения →
Надграждане →



В табличен вид настоящата методология добива следния вид:

Макрорамка

Стъпки	Ресорен служител
Инсталиране и конфигурация на софтуерната платформа (Drive-App) на сървър с дистанционен мониторинг от доставящата организация;	Доставящата организация
Тестване работата на системата с работни потребителски профили;	Доставящата организация
Обучение на служителите по разработена програма за обучение;	Дейтасист ООД и SC Networx SRL при ръководна функция на румънския партньор и функционалния експерт-организатор обучения.
Проверка степента на подготовка на служителите и провеждане на допълнително обучение (ако се налага);	Дейтасист ООД и SC Networx SRL при ръководна функция на румънския партньор и функционалния експерт-организатор обучения.
Тестово обслужване на контактни подбрани клиенти с провеждане на анкета за регистриране и анализ на пропуски и бъгове в процеса на извършване на услугата;	служителите на Дейтасист с експертни длъжности с аналитични функции
Преминаване изцяло към новия метод на работа чрез увеличаване броя на клиентите, обслужвани с иновативния метод за сметка на тези, обслужвани по досегашния;	служителите на Дейтасист с експертни длъжности с аналитични функции



ЕВРОПЕЙСКИ СЪЮЗ

Европейски Социален Фонд 2007 – 2013 Министерство на труда и социалната политика
Оперативна програма „Развитие на човешките ресурси“ BG051PO001 – 7.0.07 „Без граници –
Компонент 1“ – Фаза 2 Договор BG051PO001-7.0.07-0235-C0001 „Създаване на нови умения в
служителите на фирма „Дейтасист“ ООД за прилагане на иновативен подход при предлагане
на онлайн услуги в партньорство с фирма “SC Networx SRL” се финансира от Оперативна
програма „Развитие на човешките ресурси“ чрез Европейски Социален Фонд 2007 – 2013 и
бюджета на Р България.



Европейски социален фонд

Микрорамка

Стъпка	Ресорен служител
Клиентски казус е регистриран на Мониторинг платформата	служителите на Дейтасист с експертни длъжности с аналитични функции
Идентифициране технология и технологично оборудване на клиента	служителите на Дейтасист с експертни длъжности с аналитични функции
Използване на прототипизиращите възможности на софтуерната платформа (DriveApp) за симулиране и моделиране на клиентския казус	служителите на Дейтасист с експертни длъжности с аналитични функции
Извеждане на принципно решение	служителите на Дейтасист с експертни длъжности с аналитични функции
Имплементиране на решението на прототип	служителите на Дейтасист с експертни длъжности с аналитични функции
Извършване на дистанционен тест (имплементиране на прототипа онлайн)	служителите на Дейтасист с експертни длъжности с аналитични функции
Корекции	служителите на Дейтасист с експертни длъжности с аналитични функции
Рекапитулация на клиентския казус	служителите на Дейтасист с експертни длъжности с аналитични функции
Класифициране на казуса в базата данни на Дейтасист по вид технология, технологично оборудване и content критерии	служителите на Дейтасист с експертни длъжности с аналитични функции
Консултации, препоръки за развитие на техническата инфраструктура на клиента	служителите на Дейтасист с експертни длъжности с аналитични функции



Таблица на индикаторите за анализ

	Индикатор	Състояние
1	Клиент	
2	Служител, поел казуса	
3	Дата на клиентския казус	
4	Час на регистриране на клиентския казус	
5	Дата на решаване на клиентския казус	
6	Час на решаване на клиентския казус	
7	Технология на клиента (наименование, версия)	
8	Параметри на клиентското оборудване	
9	Клиентски казус (описание за контент анализ)	
10	Пореден номер на клиентски казус с натрупване от началото на внедряване	
11	Приложено (работещо) решение	
12	Адресирани консултации, препоръки за развитие на техническата инфраструктура на клиента	

Състоянието по индикатори от 1 до 6 и 10 се въвежда автоматично от системата за мониторинг, състоянието по останалите индикатори се отразява и попълва от служителя-експерт с аналитични функции.

Анализиране на резултатите

След акумулиране на определен брой стойности по индикатор 7. Технология на клиента, на дневна база на първо място се проследяват системни натрупвания по индикатори 10. Пореден номер на клиентски казус със натрупване, 9. Клиентски казус и 7. Технология на клиента.

На второ място се проследяват натрупвания по индикатори 2. Служител, поел казуса и 11. Приложено (работещо) решение.

Анализът на тази информация по-нататък може да позволи да се разкрият причините в съдържателен аспект, например, дали натрупванията са в резултат на присъщи на технологията на клиента характеристики, пропуски в подготовката на конкретен експерт с аналитични функции или на други фактори в процеса на обучение.

ИЗПОЛЗВАНЕ НА ПРОТОТИПИЗИРАЩИ МЕХАНИЗМИ

Средства и технологии на реализация

Приложението за прототипизиращи механизми е разделено на три основни части, които трябва да бъдат разглеждани неразделно едно от друго.

- Клиентска част
- Сървърна част
- Визуален редактор

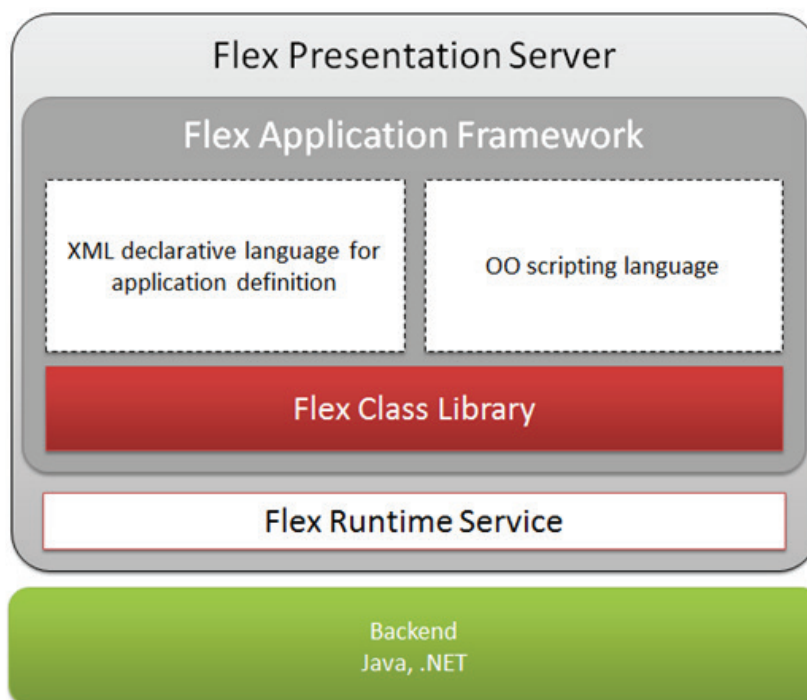
Клиентска част

Изборът на технология е съобразен с целите и задачите на приложението. Съществуват множество възможности, но със своята гъвкавост технологията Adobe Flex/AIR се явява изключително удачен избор при изграждането на модулите на приложението. Едно от основните предимства на технологията е стабилната структура на контейнера (Canvas), която позволява гъвкавост при вграждане на външни компоненти (custom components). Сложността и стабилността на контейнера е затруднена значително, имайки предвид условието приложението да работи стабилно на различни браузъри. Това прави почти невъзможно гарантирането на стабилна работа на външни компоненти в HTML среда. Самите външни компоненти трябва са съвместими с различни браузъри. Интеграцията (API interface) също е браузър зависима, което води до прекалено много пермутации и невъзможност да се гарантира работа без грешки в приложението.

Технологията е използвана както в клиентската, така и в административната част на приложението. Визията на Adobe за Flex/AIR е да създаде платформа, която да обединява хетерогенна инфраструктура от сървъри, да интегрира бизнес процеси на различни компании по стандартен начин и да предоставя на потребителя достъп до информация, която е достъпна по всяко време от всяко място и всяко устройство.

Важен е фактът, че Adobe съобразяват стратегията си с бъдещото развитие на Интернет и текущия им план включва интеграция с HTML 5. Едно от основните предимства на Adobe е платформената и браузър независимост, която се гарантира от ActionScript Virtual Machine (AVM).

Архитектура на Flex:



Сървърна част

При сървърните приложения, системата е отворена към множество сървърни платформи и да може да се интегрира с нехомогенни системи, например да обединява различни източници на информация.

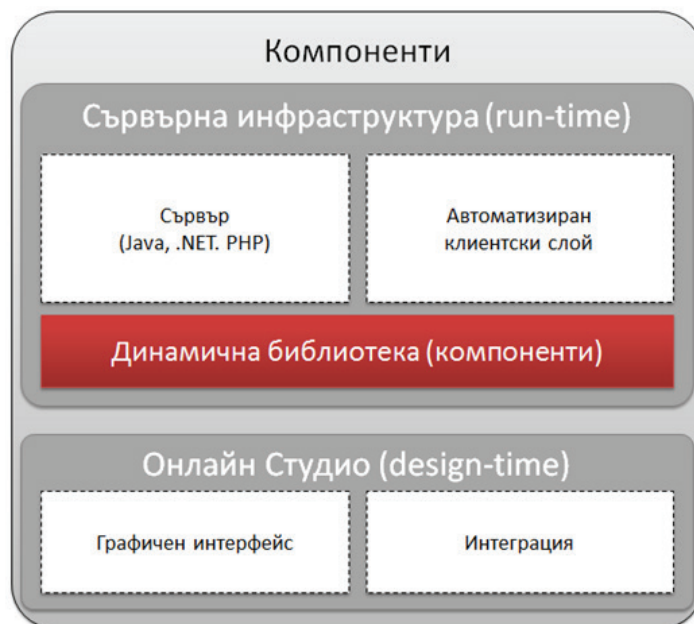
За целта платформата е разработена на трите наложени сървърни технологии: Java, .NET и PHP.

Онлайн Студио

Визуалният редактор (Online Studio) работи в браузър, но не изключва опцията за инсталирано приложение, което работи офлайн. Единствената технология, която автоматично, без допълнителна разработка, може да се превърне от онлайн в десктоп приложение, е Adobe AIR. Тази технология също така се фокусира към мобилни устройства. Много програмисти може да предпочетат средата им да има възможност за интеграция с други среди за програмиране, например Eclipse или Microsoft Studio. За целта ние е използвн Adobe AIR, който да може да работи както онлайн в браузър, така и като инсталирано приложение офлайн.

Архитектура

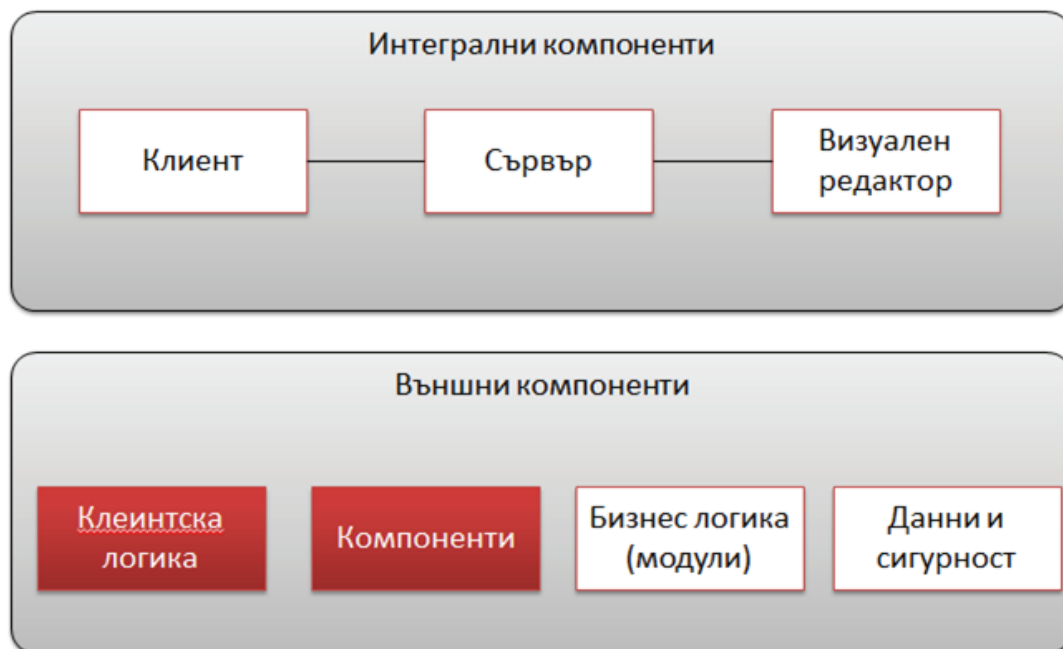
Компоненти на приложението



Архитектура на сървъра:

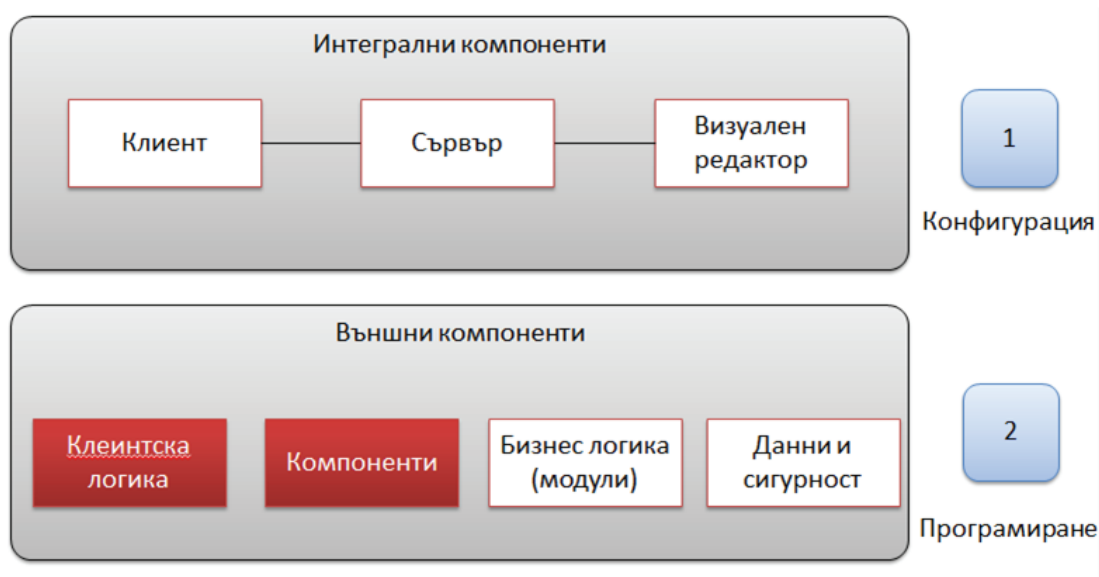


Една от основните цели на приложението е да направи разработването на приложения по-прозрачно и лесно достъпно. Driveapp предлага специална архитектура и редица инструменти, които помагат на разработчиците да разработват RIA много бързо и ефективно. Структурата на всяко DRIVEAPP приложение е прозрачна и лесна за разбиране от разработчици и потребители. Системата се състои от три основни компонента: Клиент, Сървър и Редактор.



Архитектурата е отворена към множество външни компоненти както в клиентската, така и в сървърната част.

Разработчиците могат да използват избрана от тях технология за програмиране, а в системата техните задачи са свързани с конфигуриране и дефиниране на модулите.



За да изгради приложение, програмистът има две основни задачи: да дефинира структурата на приложението и да програмира логиката на избрана от него

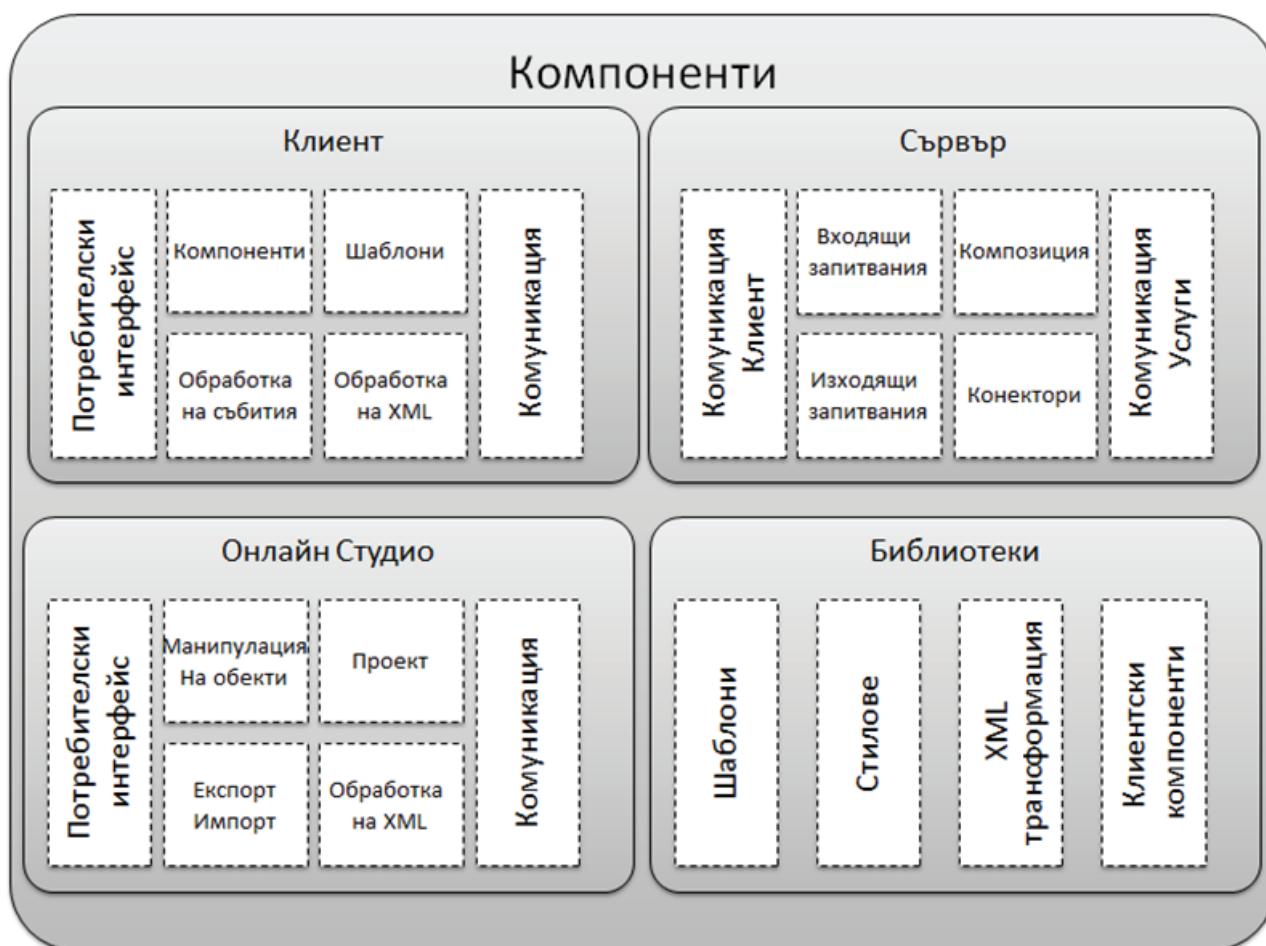
технология.

Системата поддържа шаблони за по-ефективна разработка и пълно разделение на кода от интерфейса.

Дефиницията на приложението се разделя на действия, които се описват в база данни или в XML файл.

В сърцето на системата стои декларативен XML стандарт, който описва изгледа и поведението на приложението. Нарекли сме този стандарт DRIVEAPPXML.

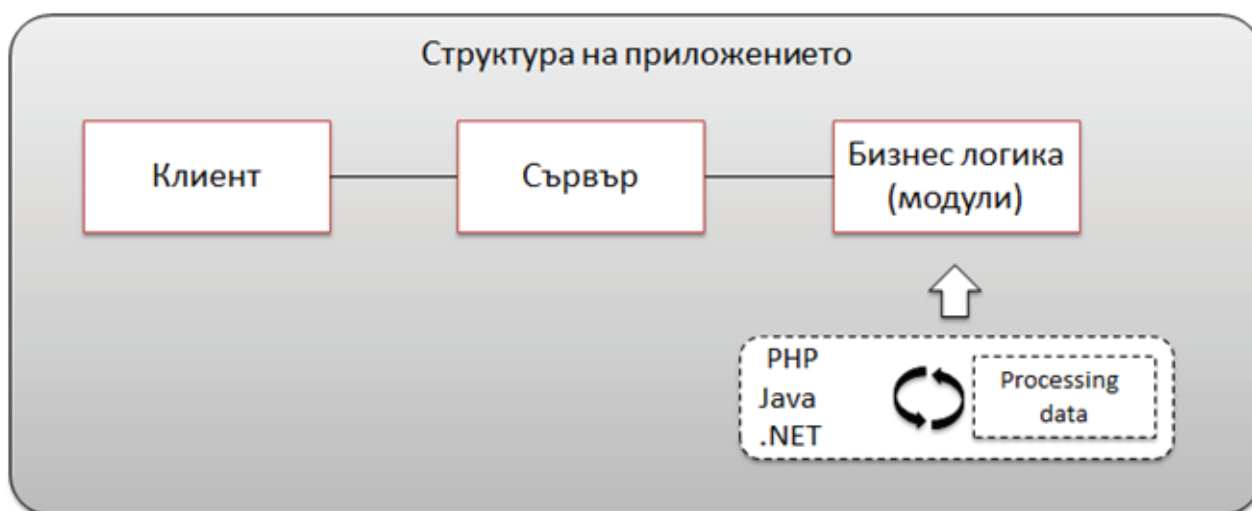
Структура



Архитектура на DRIVEAPPXML

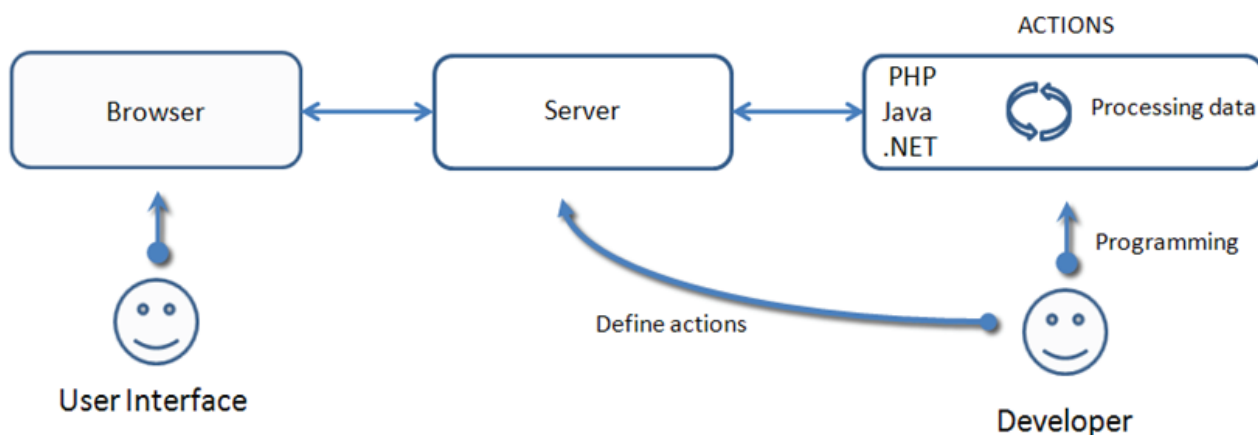
Една от основните задачи на DRIVEAPP е да направи програмирането на приложения прозрачно, ясно и лесно за поддръжка. Не винаги е възможно екипът, който е разработвал един проект, да го поддържа години наред. Това налага нуждата нови програмисти да могат да навлязат бързо в проекта. DRIVEAPP

предлага специална архитектура и инструменти, които помагат на програмистите да навлязат в съществуващия проект много бързо. Структурата на всяко DRIVEAPP приложение е прозрачна и лесна за разбиране от програмисти и потребители. Системата се състои от три основни компонента: Клиент, Сървър и Действия (бизнес логика).



Програмистът програмира на предпочитания от него език: Java, .NET, PHP или друг. Структурата на приложението и действията трябва да бъдат дефинирани в DRIVEAPP сървъра.

Overview of components



За да се програмират приложения в DRIVEAPP, има две основни задачи: дефиниране на потребителския интерфейс (шаблони) и програмиране на логиката (действия).



Бележка: Фактически потребителският интерфейс и действията могат да бъдат слети в един файл като PHP, ASPX или JSP страница.

Действията (PHP, Java, ASPX или други) се наричат събития или EVENTS (пример: “button click”) и осъществяват специфична логика (пример Login form). При натискане на бутон “Login”, действие login.php се изпълнява, за да провери потребителското удостоверение за самоличност в базата данни.

Шаблоните определят потребителския интерфейс: компоненти, стилове и събития.

Бележка: В HTML има линкове и submit-бутони във форми. В DRIVEAPP има събития като натискане на бутон (button click) или избор на обект в таблица. Но принципът е един и същ.

Жизнен цикъл на страница

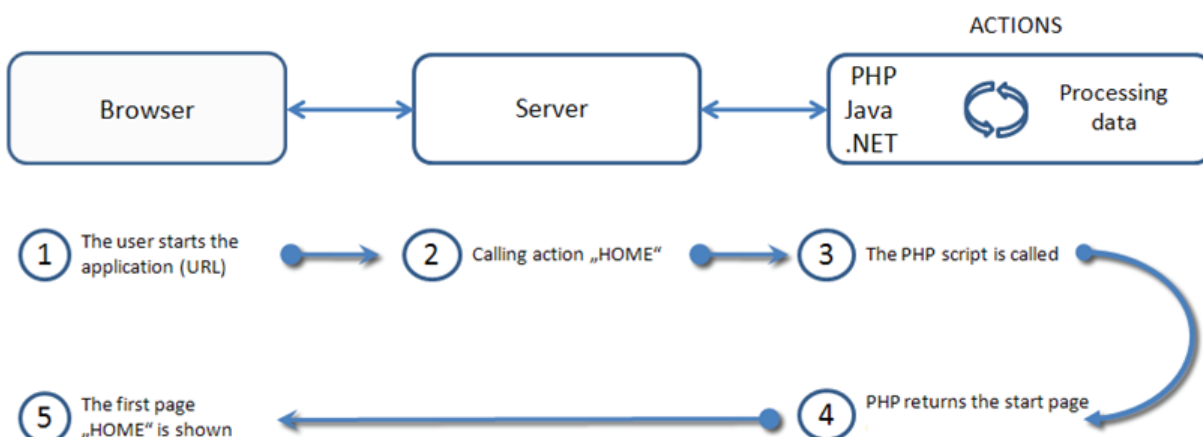
DRIVEAPP приложението е изградено от набор от действия и страници (DRIVEAPP шаблони). Подобно на уеб сайт, приложението дефинира начална страница, която се показва на потребителя, когато отваря приложението (URL).

Бележка: DRIVEAPP използва XML-базиран език на потребителския интерфейс, който е много подобен на HTML.

Някои от компонентите на потребителския интерфейс изпълняват събития (например: „натискане (click) на бутон). Тези събития се наричат действия в DRIVEAPP. Потребителят може да дефинира всички необходими действия и да ги свърже към специфични компоненти на потребителския интерфейс (бутони, картинки, таблици, ...). Това означава, че когато потребителят натисне „Бутон А“, специфично „Действие Б“ ще бъде изпълнено. Зад това действие стои обикновен сървис (PHP, .NET, Java, Ruby или друга), който управлява заявката и генерира отговор - DRIVEAPPXML документ. Това действие ще получи определени входни параметри като POST променливи (пример form data and hidden fields). Отговорът, който е генериран от действието, следва да бъде обработен и показан от клиента.

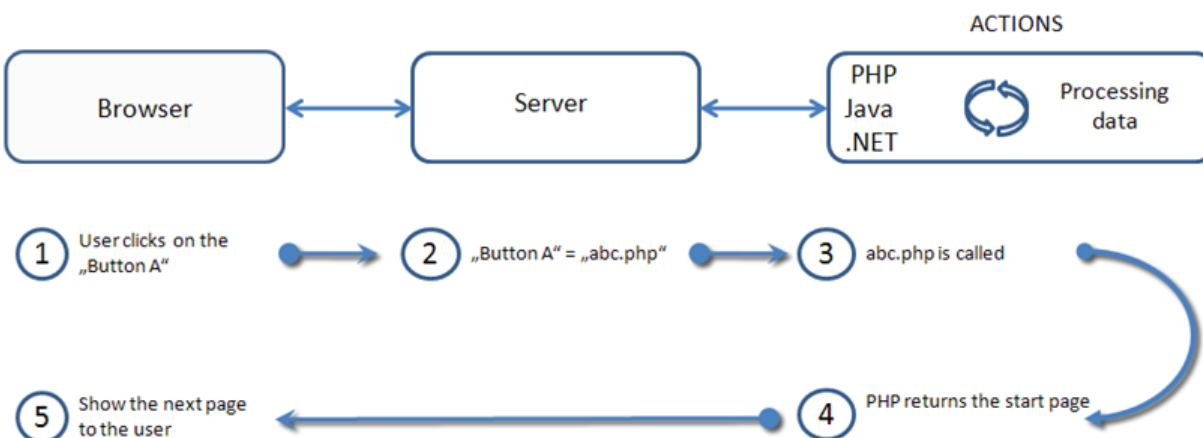
Бележка: Действията (PHP страница, .NET или Java) ще получат параметрите в POST заявка, по същия начин, по които се получават данни от HTML форма.

Page life cycle for „HOME“ – the first page of an application

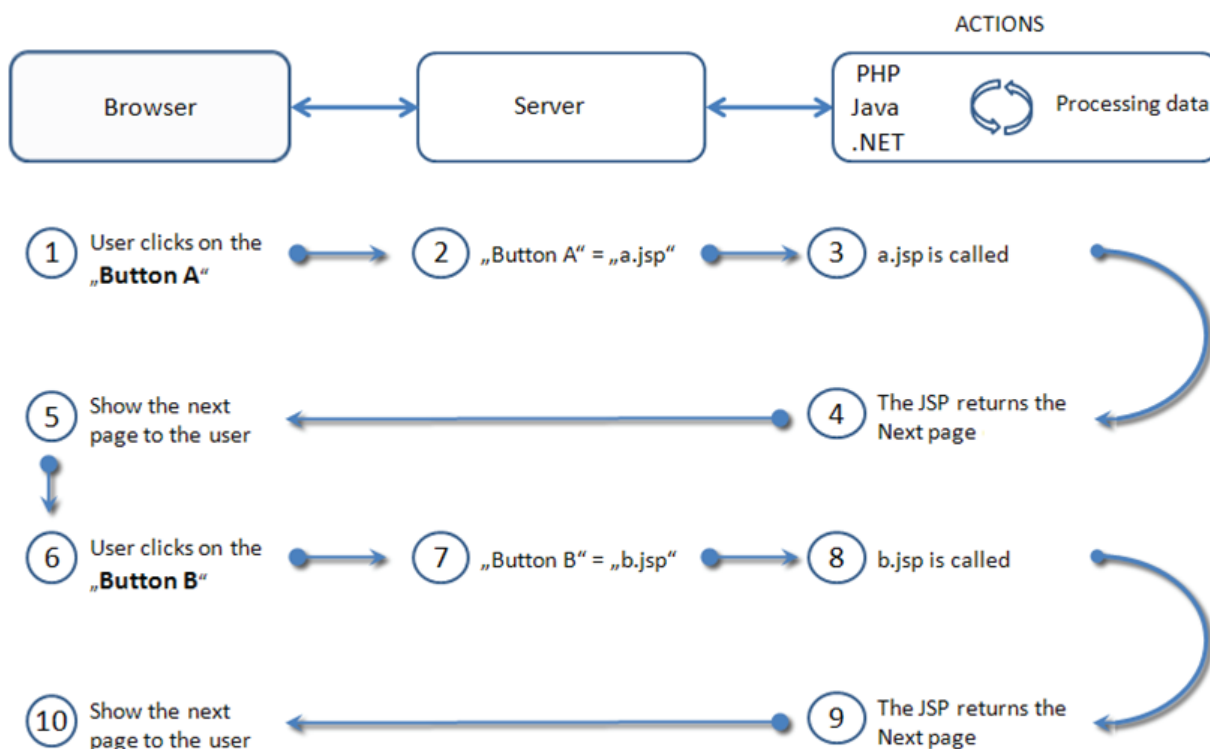


След като е показана началната страница на приложението, потребителят може да види началния екран и да започне следващото действие (например натискане на бутон). На следващата илюстрация е показан жизненият цикъл на действието.

User action => button click => show the next page



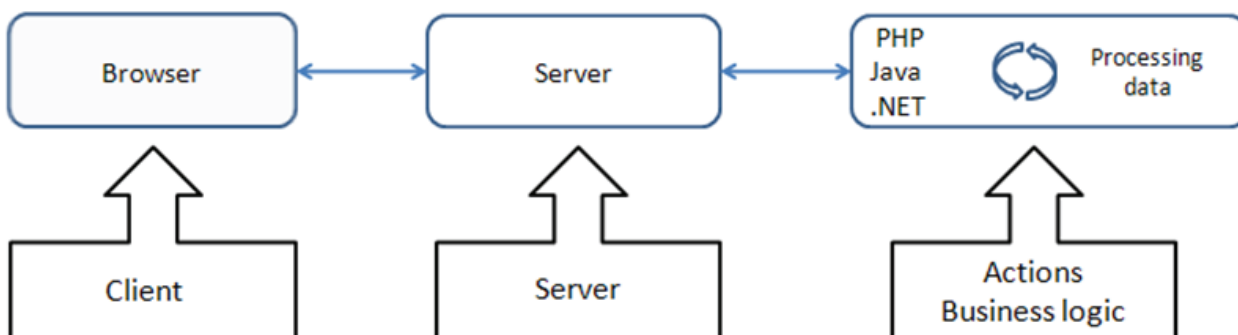
The life cycle for two button clicks



Бележка: действията могат да бъдат всяка уеб технология: PHP, Java, .NET, Ruby on Rails или друга.

Структура на компоненти

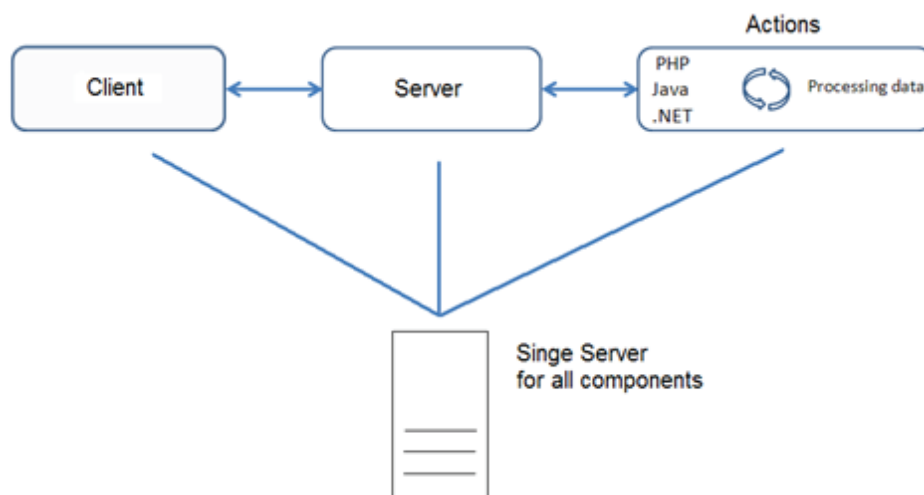
Components of the architecture



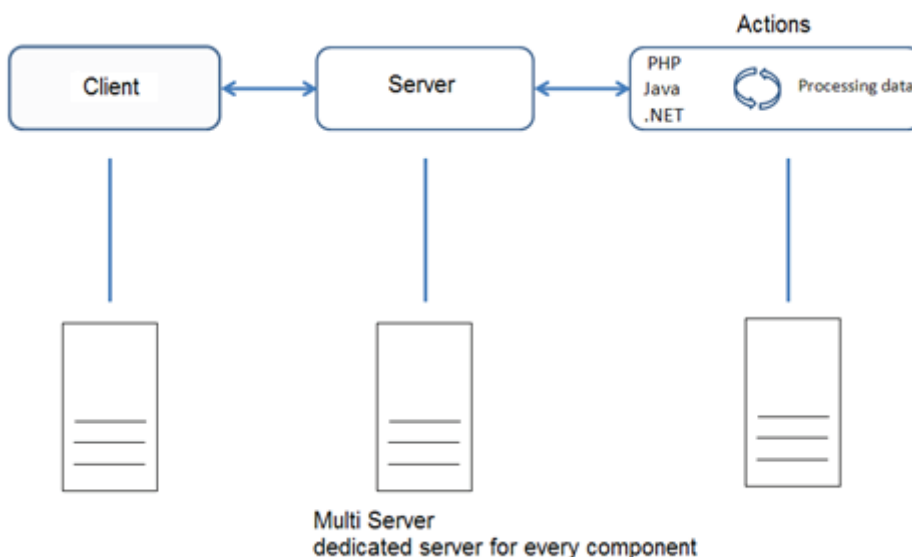
Сървърът е основната част от DRIVEAPP и е отговорен за координацията между различните части на всяко DRIVEAPP приложение. DRIVEAPP клиентът е много важен тъй като преработва DRIVEAPPXML и показва потребителския интерфейс на потребителя. Той отговаря и за обработката на събития (events). Потребителските действия са сърцевината на приложението. Тук са дефинирани логиката и екраните на приложението.

Сценарий за внедряване

Единичен сървър

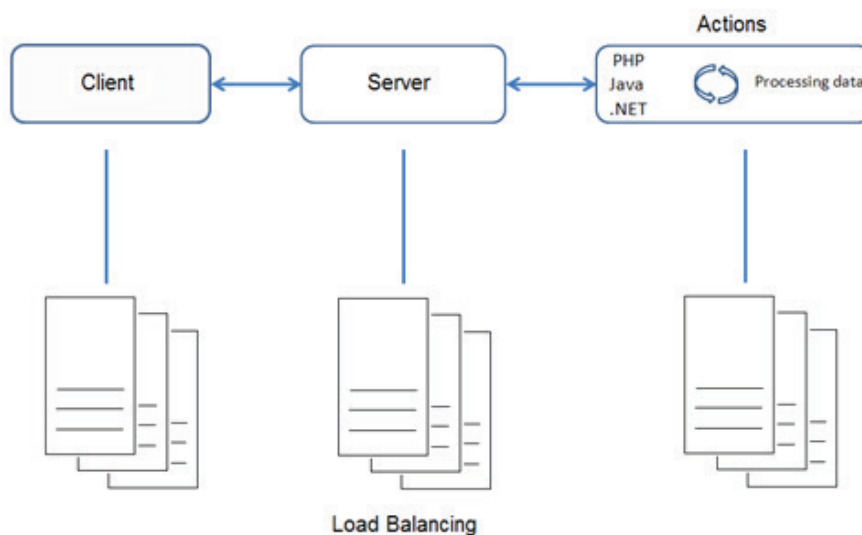


Многосървърен сценарий





Скалиране на приложението



РАЗРЕШАВАНЕ НА КОМУНИКАЦИОННИ ПРОБЛЕМИ И ТАКИВА С КОМПЮТЪРНИТЕ АКТИВИ НА КЛИЕНТА

1. Идентифицира се съобщението за грешка, които се получават и кога се получават;
2. Какви дейности предшестват съобщението за грешка;
3. Какви стъпки Клиентът вече е предприел за решаване на проблема;
4. Служител на Дейтасист ще премине през серия от стъпки по отстраняване на повредата, които ще му помогнат да диагностицира проблема
5. Ако се налага посещение на място на техник, служителят на Дейтасист ще уведоми третото лице и ще предаде допълнителни инструкции на Клиента, ако ситуацията го изисква;
6. Нива на трудност-Отговор на Дейтасист-Роля на Клиента:

Ниво на влияние	Условие	Отговор на Дейтасист	Роля на Клиента
1	Пълна загуба на критична бизнес функция, изискваща незабавен отговор	За Клиенти с мисия критична: Спешно изпращане на части и части на оторизираният служител за осигуряване на поддръжката на място паралелно с незабавно телефонно разрешаване на проблема. Бърза намеса на Мениджър поддръжка/ескалация	Осигурява 24x7 разположение на подходящ персонал за помощ
1	Пълна загуба на критична бизнес функция, изискваща незабавен отговор	За Клиенти с без мисия критична: Незабавно отстраняване на повредата по телефона. Намеса на Мениджър поддръжка/ескалации в рамките на 90 минути от установяване на връзката. Изпращане на техник на място след телефонното отстраняване на повредата	Осигурява подходящ персонал за поддържане на непрекъсната комуникация и усилия на работа.



2	Голямо влияние, но предстои работа или е на лице решение; Няма на разположение 24x7 персонал за подпомагане работата на Дейтасист	Незабавно отстраняване на повредата по телефона; наемса на Мениджъра по ескалациите в рамките на 90 минути от установяване на връзката. Изпращане на части след телефонно отстраняване на повредата и/или съобщаване на оторизирано трето лице да осигури на място човек, когато е нужно	Осигурете на разположение подходящ персонал за поддържанена Непрекъснатата комуникация и усилия за работа. Старши мениджърите трябва да са информирани и ангажирани
3	Минимално влияние върху бизнеса на Клиента	Отстраняване на повредата по телефона. Изпращане на части след телефонно отстраняване на повредата и/или съобщаване на оторизирано трето лице да осигури на място човек, когато е нужно. Статуса на случая е както се договорят взаимно страните.	Осигурява информация за място на връзка по случая и отговаря на поставените изисквания и условия от Дейтасист в рамките на 24 часа.



ПРЕПОРЪКИ ЗА РАЗВИТИЕ НА ТЕХНИЧЕСКАТА ИНФРАСТРУКТУРА НА КЛИЕНТА ПРИ ОТЧЕТЕНИ КОМУНИКАЦИОННИ ПРОБЛЕМИ И ТАКИВА С КОМПЮТЪРНИТЕ АКТИВИ НА КЛИЕНТА

В зависимост от отчетените видове проблеми в клиента могат да се дефинират препоръчителни действия свързани с преминаване към облачни услуги в три вида услуги, като този начин на работа предлага не само гъвкавост, но и доста приемлива цена. Необходимите усилия, за ефективното управление на облачна мрежа и администрирането на данните, може да бъдат облекчени чрез облачните услуги, което дава възможност на фирмите да се ангажират с други свои дейности. Малките и средни предприятия, както и новите фирми, могат да се възползват от тези удобства при използването на новите технологии. Използването на облачните услуги е страхотен начин за спестяване на разходи и време, но е много важно първо да знаете какъв вид услуга е необходима за вас.

IaaS – Infrastructure as a Service (инфраструктура като услуга)

Ползването на IaaS има много предимства. В повечето случаи клиентът получава собствен виртуален сървър, който се хоства в мрежа с реални физически сървъри, от които се разпределят ресурсите. За да се изпълняват приложенията и услугите, потребителите инсталират операционна система на сървъра, в зависимост от техните предпочитания. Ресурсите, които се разпределят на всяка индивидуална машина, могат да се контролират от потребителя и обикновено се заплаща според изразходваните ресурси и натовареност.

PaaS – Platform as a Service (платформа като услуга)

Това е решение, което се използва предимно от разработчици. Облачната услуга предоставя най-често работна среда, която включва база данни, уеб сървър и други технологии. Разработчиците използват тази готова среда, за да създадат и тестват своите продукти без да се притесняват за хардуерна съвместимост, ръчно определяне на ресурси, софтуерни инсталации и др.

SaaS – Software as a Service (софтуер като услуга)

Чрез използването на клиенти, потребителите използват софтуер, който работи в облака заедно със своята база данни. Това е обикновено продукт, който се използва при необходимост и най-често клиентите плащат на база използвани пъти. В някои случаи се плаща на абонамент принцип по срочен модел.



ОБОБЩАВАНЕ НА ТЕХНИЧЕСКИТЕ ПАРАМЕТРИ ЗА ПРЕМИНАВАНЕ КЪМ ИНОВАТИВНИЯ МЕТОД ЗА ПРЕДЛАГАНЕ НА ОНЛАЙН ПОДДРЪЖКА И ДА ДЕТАЙЛИЗИРА ЕКСПЕРТНИЯ ПЪТ, ПО КОЙТО СЛУЖИТЕЛИТЕ НА ДЕЙТАСИСТ ДА ВНЕДРЯТ НОВИЯ НАЧИН НА ПРЕДЛАГАНЕ НА УСЛУГАТА

За да се създаде ново приложение следва да се инсталират трите компонента и да се конфигурират:

- Клиент
- Сървър
- Действията (XML декларация)

Следните файлове трябва да се конфигурират:

1	Клиент	Client.html
2	Клиент	Config.xml
3	Сървър	Application.xml
4	Действия	Действие (XML) и Бизнес логика (services)

Примерна конфигурация на приложението

За да се конфигурира ново приложение с DRIVEAPP, трябва да се създаде нов конфигурационен файл. Всяко приложение има свой конфигурационен файл:

1. Име на приложението	Задава се име на проложението в секцията: <APPLICATION-NAME>
2. Пътища към приложението	Пътя към приложението следва да бъде: <SERVERINSTALLFOLDER>/DRIVEAPPServer/



3. Дефиниция на параметрите и действията на приложението

Структура на файла:

```
<?xml version="1.0" encoding="UTF-8"?>
<DRIVEAPPXML>
  <applicationparameters>
    <name>hello world</name>
    <url>http://127.0.0.1/DRIVEAPP/</url>
    <home>screen1</home>
    <templatePath>c:/xampp/myapp/template/</templatePath>
    <applicationPath>c:/xampp/myapp/php/</applicationPath>
  </applicationparameters>
  <applicationaction>
    <!-- SETTING FOR THE ACTIONS -->
    <!--ACTION 1, relative path-->
    <applicationaction>
      <applicationInsanceld>1</applicationInsanceld>
      <actionName>home</actionName>
      <actionUri>home.php</actionUri>
      <defaultaction>0</defaultaction>
      <comment />
    </applicationaction>
    <!--ACTION 2, relative path -->
    <applicationaction>
      <applicationInsanceld>1</applicationInsanceld>
      <actionName>buttonclick1</actionName>
      <actionUri>buttonclick1.php</actionUri>
      <defaultaction>0</defaultaction>
      <comment />
    </applicationaction>
    <!--ACTION 3, absolute path -->
    <applicationaction>
      <applicationInsanceld>1</applicationInsanceld>
      <actionName>buttonclick3</actionName>
      <actionUri>http://myserver.com/buttonclick3.php</actionUri>
      <defaultaction>0</defaultaction>
      <comment />
    </applicationaction>
  </applicationaction>
</DRIVEAPPXML>
```



Действия

Проектирането на приложението (връзките с програмния код) са в действията. Всяко действие се изпълнява от нормална уеб страница (PHP script, Java Servlet, JSP page, ASP.NET страница или друга), също като страница от уеб приложение. Може да се конфигурира URL-ът на всяко действие и по този начин да се използват различни сървъри и технологии за различни действия в едно и също приложение. Действията са проектирани на предпочитания от програмиста език (PHP, Java, .NET, RubyOnRails), той само трябва да конфигурира правилното URL настройките.

Примерна страница (response):

```
<DRIVEAPPXML>
  <template>
    <Label text="Hello World" />
    <Button />
  </template>
</DRIVEAPPXML>
```

Примерен PHP:

```
<?
echo "
<DRIVEAPPXML>
  <template>
    <Label text=\"Hello World\" />
    <Button />
  </template>
</DRIVEAPPXML> ";
?>
```

Примерен JSP:

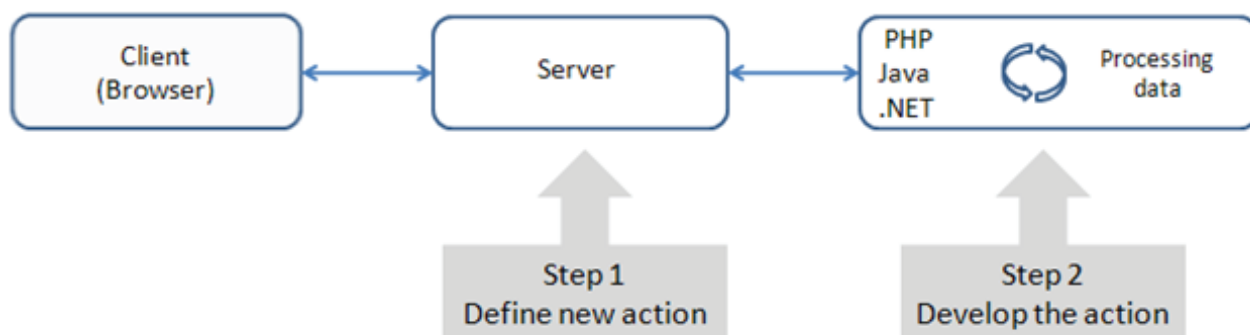
```
<%@ page contentType="text/html;charset=WINDOWS-1252"%>
<DRIVEAPPXML>
  <template>
    <Label text="<% out.println(„ Hello Oracle World“); %>" />
    <Button />
  </template>
</DRIVEAPPXML>
```

Създаване на ново действие

За да се създаде ново действие, трябва да се изпълнят следните стъпки:

Стъпка 1 – дефиниране на действието в приложението – конфигурация на xml файла на DRIVEAPP сървъра.

Стъпка 2 – създаване на URL-а (файл), където ще бъде насочено действието.



Действия и програмен код

Всяко действие се асоциира с URL, който е повикан динамично при изпълнение на действието. Този URL ще получи параметри чрез POST заявка и ще изпрати отговор в DRIVEAPPXML формат (подобно на HTML уеб страница).

Параметрите като user inputs, hidden fields и други се изпращат към действието като POST параметри. Действието може да бъде PHP page, а JSP, Servlet, .NET, Ruby on Rails или други. Всичко в DRIVEAPP е свързано с действията: всяко натискане на бутон, натискане на меню, избор на обект в таблица и други.

Пример – дефиниция на натискане на бутон:

```
<Button label="Button X" click="dispatchEvent(form1,'action1')"/>
```

В този пример ние дефинираме компонент „Бутон“ в DRIVEAPP и отговарящото събитие „натискане“. Когато потребителят натисне бутона, ще бъде генерирана POST заявка и действието „**action1**“ ще бъде повикано.

Действието „**action1**“ е описано в конфигурацията на приложението и се асоциира с URL.



Структура на страница DRIVEAPPXML

Въведение

Както е описано в предишните секции, действията са извиквани чрез POST заявка и следва да изпратят отговор към DRIVEAPP сървър. Този отговор трябва да е в XML формат и да спазва стриктно структурата на DRIVEAPPXML.

Структурата на DRIVEAPP страница се състои от 3 секции:

1. Потребителски интерфейс	Дефиниция на оформлението на екрана
2. Данни	Данни на екрана
3. Команди	Контролиращи компоненти

Всяка секция ще бъде описана по отделно в определена глава:

Бележка: Не е задължително да се дефинират всички 3 компонента. Нулев отговор <DRIVEAPPXML/> също е валиден.

Основите

Всяка DRIVEAPP страница има предварително дефинирана XML структура. Основният компонент е <DRIVEAPPXML>.

Три основни секции са налични с root-tag: <template>, <templatedata> и <commands>.

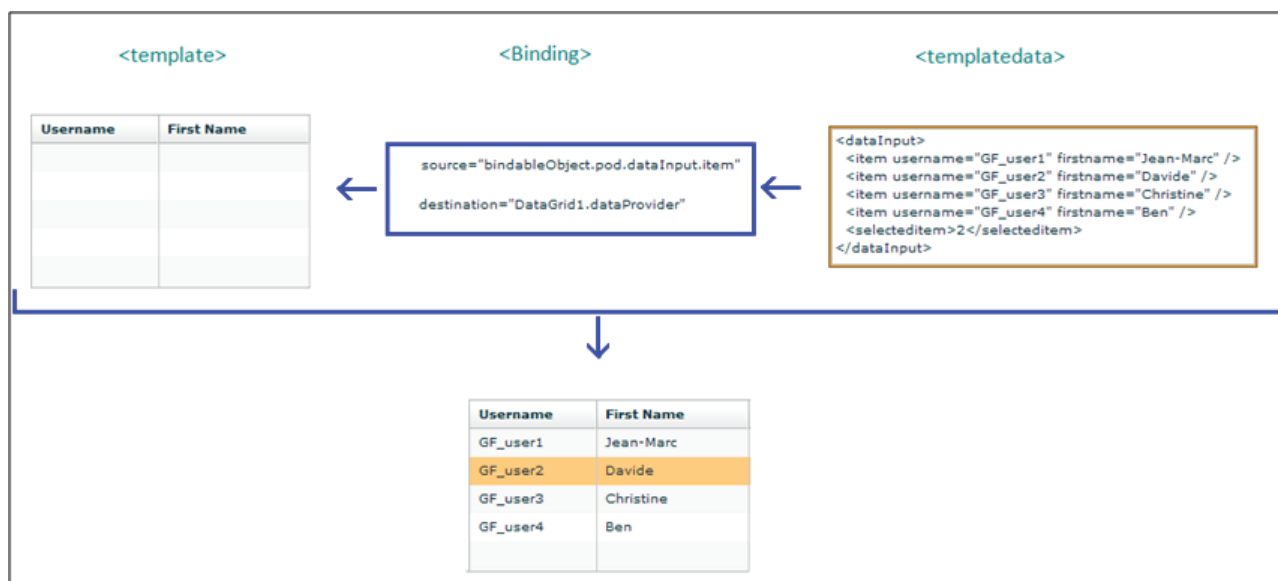
Пример за DRIVEAPP страница:

```
<DRIVEAPPXML>
  <template>
    1. USER INTERFACE COMPONENTS
  </template>
  <templatedata>
    2. DATA-XML
  </templatedata>
  <commands>
    3. COMMANDS
  </commands>
</DRIVEAPPXML>
```

Детайли за трите основни секции, които изграждат потребителския интерфейс:

<code><template></code>	In the section <code><template></code> the user interface is defined. This are horizontal boxes, vertical boxes, buttons, images, text input fields, DataGrids and other.
<code><Binding></code>	You can use the command <code><Binding></code> to connect user interface components with data from templatedata. The commands <code><Binding></code> should be placed within the tag <code><template></code> Example: <code><Binding source="templatedata.customerslist.item" destination="customersDataGrid.dataProvider"/></code>
<code><templatedata></code>	In the section <code><templatedata></code> the data in xml format is defined. The components defined in <code><template></code> can be mapped (binding) to this data. This makes separation between layout and data possible and very comfortable. For example a DataGrid with <code>id="customers"</code> links to data <code>templatedata.customerslist</code>

Свързване на данните към шаблона:





1- Потребителски интерфейс

Потребителският интерфейс, както името предполага, се състои от потребителски интерфейс, компоненти като бутони, картинки, етикети, таблици, текстови полета и други. Може да се направи и свързване между тези компоненти и данните, определени във втора секция.

Шаблони

Тагът `<template>` (шаблон) е основен таг в DRIVEAPP страница. Той се използва, за да дефинира оформлението на страницата, където се определят контейнерите и компонентите, за да се изгради оформлението на потребителския интерфейс.

Пример:

```
<DRIVEAPPXML>
  <template>
    <Button/>
    <Label/>
    <DataGrid/>
  </template>
</DRIVEAPPXML>
```

`<template>` тагове могат да бъдат използвани като вложени елементи в друга страница:

Пример:

```
<DRIVEAPPXML>
  <template>
    <Label/>
    <DataGrid/>
  </template>
  <template>
    <Button/>
    <Button/>
    <Button/>
  </template>
</DRIVEAPPXML>
```

Свързване на данни

Концепцията **за разделението между компоненти и данни** е добре позната практика в програмирането на приложения. То позволява ясно да бъде разделена визуализацията (потребителския интерфейс) от логиката и достъпа до база данни. То също позволява да бъдат обновени само данните – свързването ще обнови елементите от само себе си.

Структурата на тага `<templatedata>` може да бъде всеки валиден XML с всичките данни за екрана, които могат да бъдат адресирани с `<Binding>` команда.

В командата `<Binding>` може да се свържат компонентите на потребителския интерфейс от `<template>` към данните в `<templatedata>`. Източникът на свойствата насочва към `<templatedata>`, докато местоположението на свойствата насочва към `<template>`.

Пример:

```
<Binding source="templatedata.mydata" destination="txtInput.text"/>
```

Това означава, че с таг `<templatedata>`, таг, който се нарича `<mydata>`, трябва да съществува. Стойността на `<mydata>` ще бъде свързана с компонент с идентификационно име (ID) `"txtInput"` и неговите свойства `"text"`.

Пример за `<templatedata>`:

```
<templatedata>  
  <mydata> this is my text </mydata>  
</templatedata>
```

Свързването е също много полезно, когато се обновяват компоненти или техните свойства и данни на екрана (без да има нужда да се презареждат компонентите на екрана, подобно на AJAX). Например, след като бутонът „Търсене“ е натиснат, данните от таблицата ще бъдат обновени и резултатите от търсенето ще бъдат показани. Таблицата е свързана с таг в `<templatedata>` и в промяна на данните на компонента, компонентът е автоматично обновен. В този случай, след като бутонът „Търсене“ е натиснат, отговорът следва да се върне само към секция `<templatedata>` с новите данни.



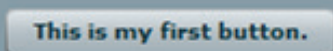
Пример `<templatedata>` връща само данни:

```
<DRIVEAPPXML>  
<templatedata>  
  <musician>  
    <name>Bill Cosby</name>  
    <name>Tom Selleck</name>  
    <name>Louis Armstrong</name>  
  </musician>  
</templatedata>  
</DRIVEAPPXML>
```

Command: `<Button id="Button1" />`

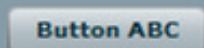


Without binding of the property label



With binding of the property label

Command: `<Button label="Button ABC"/>`



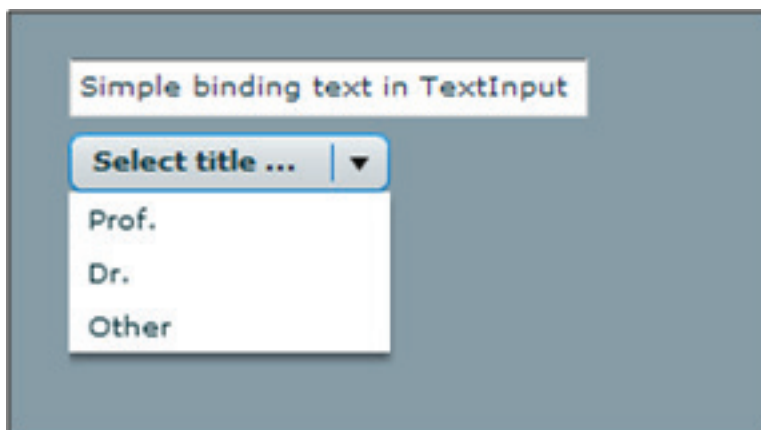
Without binding. The property label can be defined directly within the component.



Пример:

```
<DRIVEAPPXML>
  <template>
    <VBox id="dynamic" height="100%">
      <TextInput id='txtInput' />
      <Binding source="templatedata.sourceTextInput" destination="txtInput.text"/>
      <ComboBox id='cmbInput' prompt='Select title ...' labelField='name' />
      <Binding source="templatedata.sourceTitle.item" destination="cmbInput.dataProvider"/>
    </VBox>
  </template>
  <templatedata>
    <sourceTextInput>Simple binding text in TextInput</sourceTextInput>
    <sourceTitle>
      <item>
        <name>Prof.</name>
        <value>0</value>
      </item>
      <item>
        <name>Dr.</name>
        <value>1</value>
      </item>
      <item>
        <name>Other</name>
        <value>2</value>
      </item>
    </sourceTitle>
  </templatedata>
</DRIVEAPPXML>
```

Снимка на горния пример:

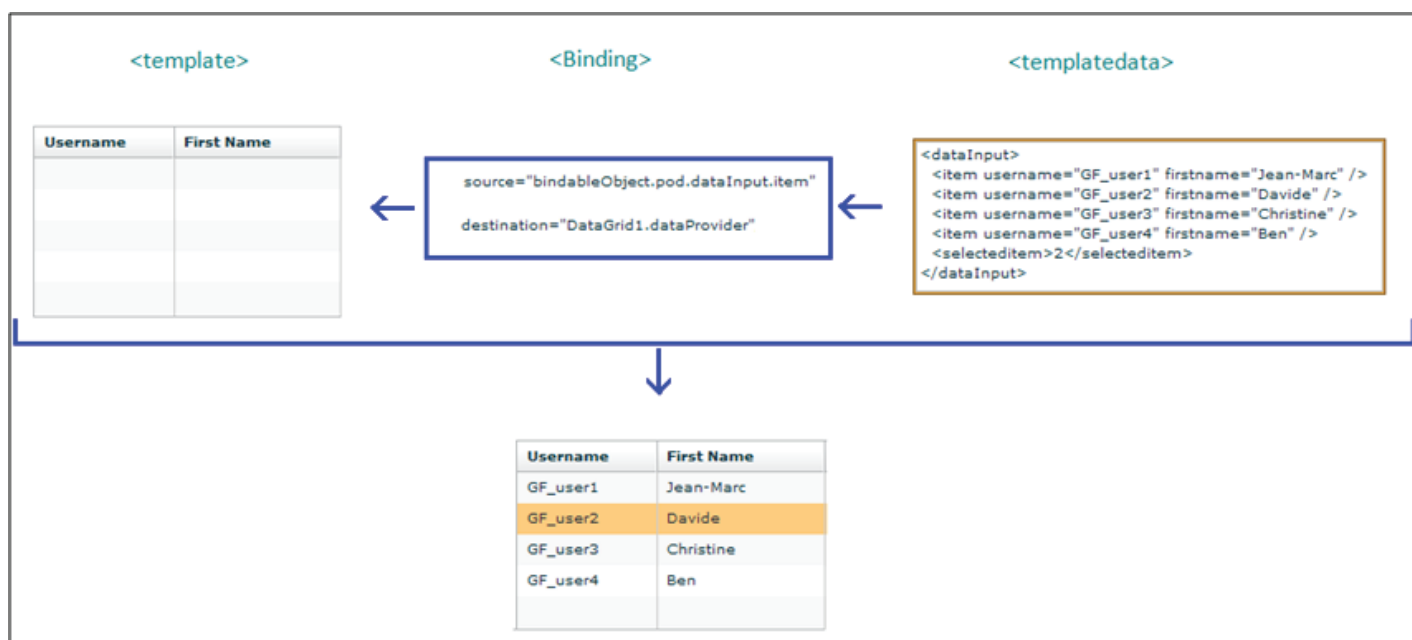


Бележка: Можете да свържете не само данни, но също така и всякакви други свойства на елемент: например цвят, етикет и други.

2. Данни XML

Тагът `<templatedata>` определя специфични данни или свойства на компонентите на потребителския интерфейс, дефинирани в `<template>`. За пример може да се дефинира Таблица, данните, заредени в таблицата, са дефинирани в `<templatedata>` секцията. Тагът `<templatedata>` може да има много данни от различни компоненти на екрана. С таг `<Binding>` може да определим кои данни за кой компонент се използват.

Пример таблица (компонент+свързване+данни):



Пример `<templatedata>`:

```
<DRIVEAPPXML>
  <templatedata>
    <musician>
      <name>Bill Cosby</name>
      <name>Tom Selleck</name>
      <name>Louis Armstrong</name>
    </musician>
  </templatedata>
</DRIVEAPPXML>
```

Бележка: с `<templatedata>` може да се използва всяка валидна XML структура.



Ако компонент е свързан с данни в `<templatedata>`, всеки следващ отговор от данните може да бъде презареден и компонентът да бъде показан отново автоматично (data binding). Компонентите на потребителския интерфейс са свързани с „пътека“ в секцията с данни и шаблони: например `templatedata.customers.item`. Ако стойността на тези данни е променена (с отговор от действие), свързването ще обнови всички компоненти, свързани с тази стойност. Трябва да се избегне използването на едни и същи пътеки, защото може да доведе до нежелани резултати.

Бележка: За да се избегнат проблеми със свързването, следва да се използват уникални IDs (имена), като например SCREENNAME_SECTIONNAME_COMPONENTID. Това би направило приложението по-ясно и структурирано.

Добър пример за templatedata:

```
<templatedata>
  <customerlist_label>....</customerlist_label>
  <customerlist_name>....</customerlist_name>
  <customerlist_surname>....</customerlist_surname>
</templatedata>
```

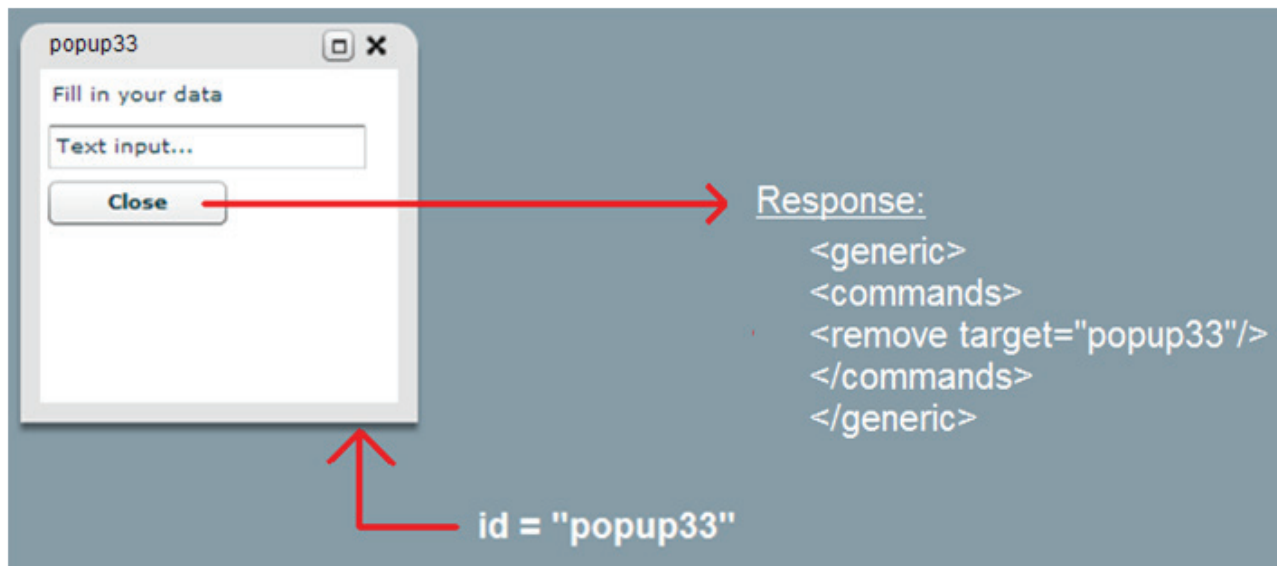
3 - Команди

Командите в DRIVEAPP се използват за анимации, показване и премахване на компоненти от екрана.

Две са командите за скриване на компоненти:

1. Премахване на един компонент
2. Премахване на всички компоненти = чист екран

Например при натискане на бутона „Отмяна“ следва да се премахне изскачащият прозорец. ID-то на изскачащия прозорец е „rorip33“, при натискане на бутона действието ще изпрати отговор с команда премахни „rorip33“.



Премахване на един компонент от екрана

Премахване на един компонент:

```
<DRIVEAPPXML>
  <commands>
    <remove target="ID-COMPONENT"/>
  </commands>
</DRIVEAPPXML>
```

В този случай, компонентът с **ID-COMPONENT** ще бъде премахнат.

Премахване на всички компоненти на екрана

Синтаксисът за премахване на всички компоненти от екрана е следният:

```
<DRIVEAPPXML>
  <commands>
    <removeAll/>
  </commands>
</DRIVEAPPXML>
```

Очевидно командата "remove all" (премахни всички) не може да бъде използвана самостоятелно, без да се показват новите компоненти на екрана, защото всичко на екрана би се изтрило и оставило потребителя с празен екран. Тази команда се използва в края на определянето на екрана, когато следва да бъде показан празен заден план.

Пример:

```
<DRIVEAPPXML>
  <template>
    <VBox width="100%" height="100%" horizontalAlign="center">
      <Label text="Request form"/>
      <HelpLinkButton label="Take an Action" />
    </VBox>
  </template>
  <commands>
    <removeAll/>
  </commands>
</DRIVEAPPXML>
```

Снимка на екрана (След „remove all“ само нови компоненти от `<template>` са показани):



Структура на страници (детайли)

Вече се запознахме с основната структура на страницата DRIVEAPPXML. В тази секция ще бъдат изложени уникалните характеристики на някои от най-важните свойства.

Цел (AJAX-стил)

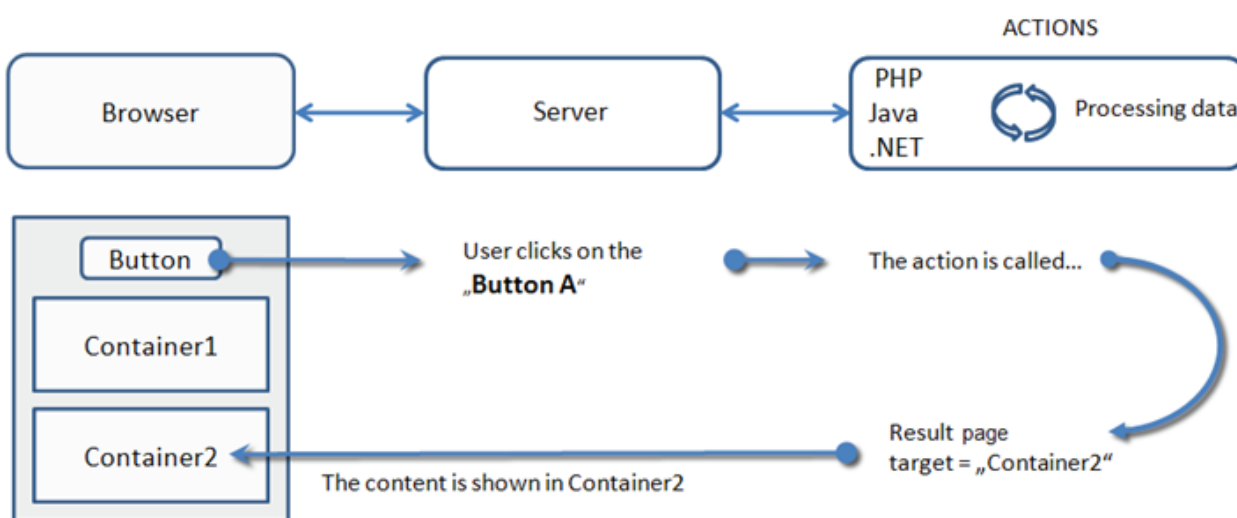
Свойството `target` може да бъде използвано с `<template>` или `<templatedata>`. То може да специфицира целеви контейнер, където съдържанието следва да бъде показано. По този начин може удобно да се презаредят само части от екрана, без да се използва допълнително кодиране.

Синтаксис:

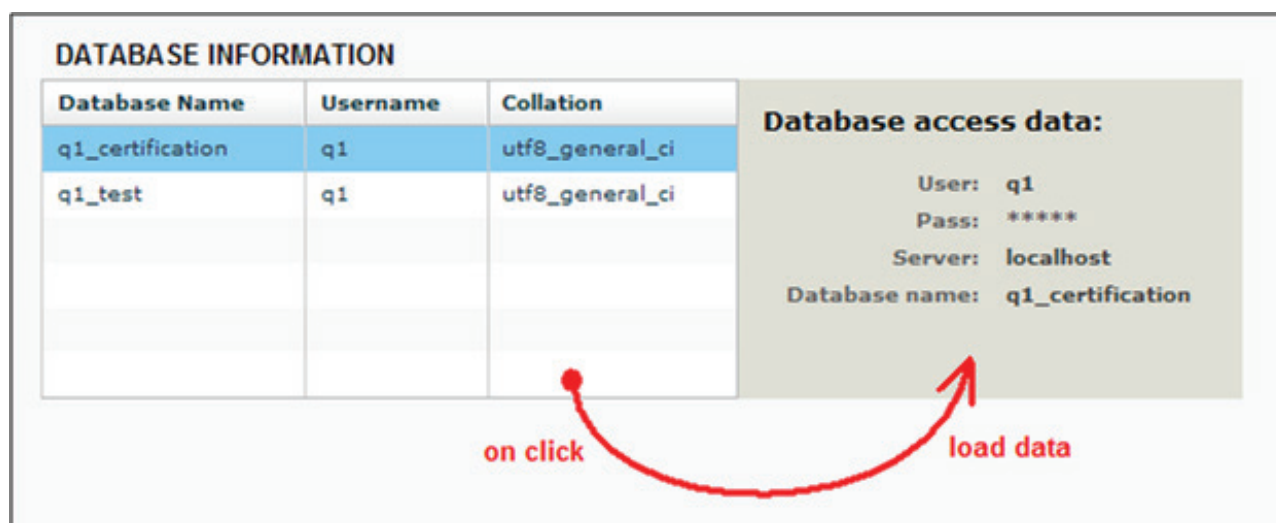
```
<DRIVEAPPXML>
  <template target="ID-TARGETCONTAINER">
    ...
```

Основна схема:

Target = „Container2“



Пример: Зареждане на детали след натискане на ред:





Сценарий: Потребителят избира ред в таблицата, действието е повикано (изпратена е заявка) и детайлите за този ред са показани до таблицата в предварително дефиниран контейнер – без нуждата да бъде показан отново целият екран.

В този случай натискането на ред на таблицата ще повика действие (заявка), която ще върне необходимите компоненти със специфична цел (отговор). Всеки път, когато потребителят натисне ред, в дясната част на екрана ще бъде заменено ново съдържание, без да се презарежда целия екран.

Бележка: Цел (target) се използва само когато искате да замените части от екрана.

Бележка: Целевият контейнер се премахва и заменя с нов. Новият контейнер следва да има същото ID като предишния. Тъй като потребителят може да иска да предприеме същото действие отново (като например избиране на различен ред в таблицата), следва целевото ID да съществува.

Външни шаблони

DRIVEAPP е изграден върху разделението между потребителския интерфейс и логиката. В DRIVEAPP може да се зареди шаблон (потребителски интерфейс) от външни шаблонни файлове. Това може да бъде много полезно за организацията на програмния код.

Например, може да се дефинира меню на приложението във външен файл, който да се включи към всеки екран.

За да се зареди съдържанието от външен файл може да се използва `<template name=„abc“/>` където „abc“ е името на външното съдържание. Местоположението на шаблонните файлове се намира в определените в приложението свойства.

Използване на външни шаблонни файлове:

```
<DRIVEAPPXML>
  <template name=„abc“/>
  <!--The contend will be loaded from abc.mxml located in the templates folder -->
  <templatedata>
    ...DATA-XML...
  </templatedata>
</DRIVEAPPXML>
```



Бележка: Всички компоненти са определени в отделен файл. Abs.tmxl е името на файла, който ще бъде зареден.

Може да се използват `<template>` тагове също с други `<template>` тагове.
Пример:

```
<DRIVEAPPXML>  
  <template>  
    <VBox>  
      <template name="menu"/>  
      <Button/>  
      <DataGrid/>  
    </VBox>  
  </template>  
</DRIVEAPPXML>
```

Добрите практики показват, че повече от екраните могат да бъдат външни статични шаблони и действието се връща с името на шаблона `<template name="screenX"/>`, `<templatedata>` и `<commands>`.

Използване на променливи във външните шаблони

Много е полезно да се използват променливи за пътеките на картинки и други често употребявани поредици в шаблоните. Използването на променливи в шаблоните е лесно.

Пример:

```
<Image source="{img_path}/logo.jpg"/>
```

Дефиницията на константите е в конфигурационния xml файл на приложението.

Пример:

```
<applicationconstant>  
  <name>img_path</name>  
  <value>http://127.0.0.1/images/</value>  
</applicationconstant>
```

Покажи/Скрий

Всеки компонент в DRIVEAPP има две свойства, които определят неговото показване (явен или скрит): **includeInLayout** и **visible**. В тази секция ще разгледаме тези две важни свойства:

1. includeInLayout – Включен в оформлението

Стойности: „true“ или „false“

Описание: Специфицира дали този компонент да бъде включен или не в оформлението на контейнера майка.

2. visible - видим

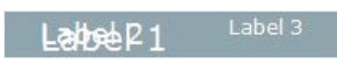
Стойности: „true“ или „false“

Описание: Контролира видимостта на този компонент

Пример:



Съществуват 4 различни оформления в зависимост от стойностите **visible** и **includeInLayout** на етикет 1.

Visible	includeInLayout	Screenshot
TRUE	TRUE	
FALSE	TRUE	
TRUE	FALSE	
FALSE	FALSE	



Ако искате напълно да скриете даден компонент, следва да определите и двете стойности **`visible = "false"`** и **`includeInLayout="false"`**. Ако определите само едната **`visible="false"`**, компонентът няма да е видим, но ще бъде в позиция. В горепосочения пример празното място, където компонентът следва да бъде показан, ще бъде запазено.

Пример:

```
<Label text="MyLabel" includeInLayout="true" visible="false"/>
```

Използване на Покази/Скрий компонент със свързване на данни:

```
<DRIVEAPPXML>  
<template>  
  <Label id="mylabel" text="Label 1" color="white" fontSize="22"/>  
  <Binding source="templatedata.label1 visible"  
    destination="mylabel.includeInLayout" />  
</template>  
<templatedata>  
  <label1 visible>true</label1 visible>  
</templatedata>  
</DRIVEAPPXML>
```

Това е полезно, тъй като лесно може да се показват или скриват компоненти с натискане на бутон или други събития чрез използване на свързване и определяне на отговарящите `<templatedata>` в действието.

Вградени ресурси

В някои ситуации е полезно да се вгради картинка към определен компонент, например икона на бутон или стрелка към плъзгач на HSlider компонент.

Пример - HSlider:

```
<HSlider />  
<HSlider thumbSkin="@Embed(source='http://127.0.0.1/icon.png')" />
```

Снимка:



Пример- Button:

```
<Button label="Button" />  
<Button icon="@Embed(source='http://127.0.0.1/button_add.png')" />  
<Button icon="@Embed(source='http://127.0.0.1/button_add2.png')" />
```

Снимка: бутон с вградена икона:



Бутони и действия

Действия

На всеки компонент в DRIVEAPP потребителят може да определи натискане/клик "click" (или "doubleClick") събитие. Други събития също са поддържани от други компоненти, като например "selectedItem" в таблица.

Най-често използван е "click" за бутон. Когато потребителят натисне бутона, действието ще бъде повикано (заявка), която ще изпълни определена логика и ще върне резултат (отговор). Резултатът следва да бъде валидна структура на страница. Обикновено програмирането на приложение започва с определяне на първия екран на приложението "home"/начална страница. Това може да бъде регистрационен екран /Login screen/. На първоначалната страница ще



имаме бутони, картинки и други компоненти с дефинирани „действия“. Тъй като нашето приложение най-вероятно не е приложение с единствен статичен екран, ние искаме да дефинираме “clickable” компоненти. След като потребителят натисне върху компонент с предварително определени събития, действието е повикано (**изпратена е заявка**). Тази заявка отива до URL-а (PHP script, Java page, ASPX или друго), след това се генерира отговор, който също следва да бъде определен на следващ екран на приложението. Този отговор следва да бъде валидна страница.

Пример за бутон с click събитие:

```
<Button click="dispatchEvent( form1 ,buttonclick)" />
```

Функцията `dispatchEvent(<formID>,<actionName>)` е специална функция, която определя кое действие следва да бъде повикано и какви данни да бъдат изпратени. Параметрите тук са много важни:

Parameter	
<code><formID></code>	<code>formID</code> -то е идентификационен код на контейнер или форма. Този контейнер може да съдържа оформящи компоненти като <code>TextInput</code> , <code>DataGrid</code> , <code>ComboBox</code> или всеки друг оформящ компонент. Когато заявката е генерирана, всички данни от оформящите компоненти ще бъдат изпратени с действие (заявка) и данните могат да се обработят в кода. Оформящ контейнер може да бъде всеки контейнер <code>HBox</code> , <code>VBox</code> или друг.
<code><actionName></code>	Името на действието, което ще бъде повикано. Виж секция Действия

Ако искаме да дефинираме бутон, който потребителят да натиска, може да го направим чрез определяне на “click” събитие на бутона.

Пример:

```
<Button label="Search" click="dispatchEvent(form1,,action33)"/>
```

Но ако има нужда от данни, въведени от потребителя (по примера по-долу с търсенето) да бъдат изпратени до действието със заявка, следва да специфицираме контейнер, където се намират компоненти със заявка.

Снимка:



В горния пример контейнерът, където се намира търсенето, TextInput, следва да бъде дефиниран като параметър при определянето на натискане на бутон.

Форми

Формата е определена като колекция от различни интерактивни компоненти. Интерактивен компонент е всеки компонент, който позволява на потребителя да въвежда данни или да прави избор. Пример:

<CheckBox>	☒ Simple CheckBox										
<ComboBox>	one ▼										
<Slider>	<input type="range"/>										
<NumericStepper>	20 ▲▼										
<TextInput>	Input text..										
<DataGrid>	<table border="1"><thead><tr><th>First Name</th><th>Second Name</th></tr></thead><tbody><tr><td>Jean-Marc</td><td>Agator</td></tr><tr><td>Davide</td><td>Damosso</td></tr><tr><td>Christine</td><td>Mason</td></tr><tr><td>Ben</td><td>Mayo</td></tr></tbody></table>	First Name	Second Name	Jean-Marc	Agator	Davide	Damosso	Christine	Mason	Ben	Mayo
First Name	Second Name										
Jean-Marc	Agator										
Davide	Damosso										
Christine	Mason										
Ben	Mayo										

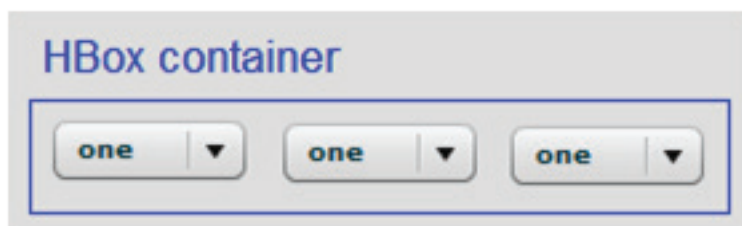
Бележка: Таблицата може да бъде част от форма. Избраният ред ще бъде изпратен до сървъра.

Примери за **неинтерактивни компоненти** са картинките, етикетите и други компоненти, където потребителят не може да направи нищо. Интерактивните компоненти са поместени в контейнери.

Пример HBox:

```
<HBox id="form1">  
  <ComboBox id="one"/>  
  <ComboBox id="two"/>  
  <ComboBox id="three"/>  
</HBox>
```

Снимка:



В горния пример контейнерът HBox с id "form1" е интерактивен контейнер. Във функцията `dispatchEvent(form1, actionname)`, първият параметър `formId` очаква id-то на контейнера.

Бутонът „Изпрати“/ „Send“ може да има click събитие, определено с функцията `dispatchEvent(form1, actionId)`

Пример HBox като интерактивен контейнер:

```
<HBox id="form1">  
  <ComboBox id="one"/>  
  <ComboBox id="two"/>  
  <ComboBox id="three"/>  
</HBox>  
<Button label="Send" click="dispatchEvent(form1, anyaction)"/>
```

Когато потребителят натисне “Send” dispatchEvent() ще изпрати данни от контейнера:



Скрити полета

Скритите полета, както името предполага, са полета, които се използват за съхранение на специфични за екрана данни, но не са показани на потребителския интерфейс, също както в HTML `<input type="hidden">`. Обикновено `<TextInput>` се използва като скрито поле. За да се направи скритото поле `<TextInput>` нормално, следва да специфицираме неговите опции за видимост: `includeInLayout="false"` и `visible="false"`.

Пример:

```
<TextInput id="comment_id_case" text="myData" includeInLayout="false" visible="false"/>
```

В този случай `<TextInput>` няма да бъде видим на екрана, но ще бъде част от контейнера и данните ще бъдат изпратени със заявка. Това е полезно, когато се отваря изскачащ прозорец, за да се редактира потребител и да се сложи ID на потребителя в скрития `<TextInput>`.

Списък на всички интерактивни компоненти:

Form-components	Description
AdvancedDataGrid	The AdvancedDataGrid control extends the capabilities of the standard DataGrid control with: sorting by multiple columns displaying hierarchical and grouped data
Button	Common rectangular button. Label or icon is supported.



CheckBox	Commonly used graphical control that can contain a check mark.
ColorPicker	The ColorPicker control lets users select a color from a drop-down swatch panel (palette).
ComboBox	The ComboBox control contains a drop-down list from which the user can select a single value.
DataGrid	Multi column display of data.
DateChooser	Calender from which the user can choose a date.
DateField	The DateField control is a text field that shows the date with a calendar icon on its right side.
HSlider/VSlider	You can use the slider controls to select a value by moving a slider thumb between the end points of the slider track.
LinkButton	The LinkButton control is a borderless Button control whose contents are highlighted when a user moves the mouse over it.
List	The List control displays a vertical list of items. The user can select single or multiple values.
NumericStepper	The NumericStepper control lets the user select a number from an ordered set.
RadioButton	The RadioButton control lets the user make a single choice within a set of values.
RadioButton-Group	By using the property groupName="cardtype" you can create a group of RadioButtons. Within this group only one value can be selected.
RichTextEditor	The RichTextEditor control lets users enter, edit, and format text.
TextArea	The TextArea control is a multiline text field.
TextInput	The TextInput control is a single-line text field.
Tree	The Tree control allows you to show hierarchical data arranged as an expandable tree.

Прозорец при зареждане на данни

Функцията `dispatchEvent( <formID>, <actionName> )` е трети параметър, който определя дали зареждането на изскачащ прозорец да бъде показано или не. По подразбиране стойността е валидна и ако не зададем трети параметър, изскачащият прозорец ще бъде показан.

Пример:

```
<Button label="Search" click="dispatchEvent(form1,,action33', false)"/>
```

Зареждащ се прозорец:

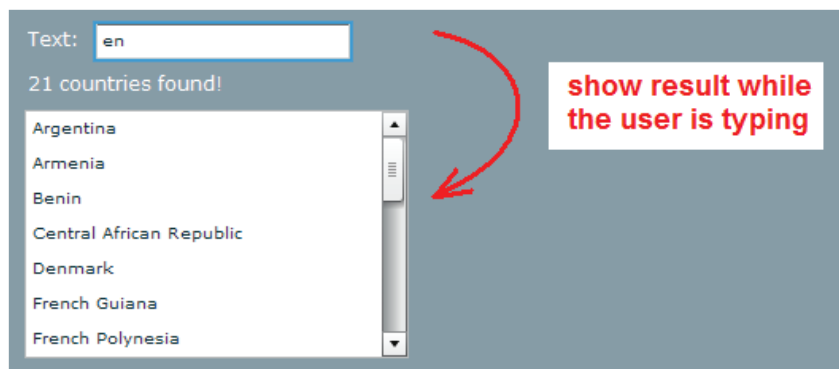


Бележка: Показването на зареждащ се изскачащ прозорец е препоръчително. Зареждащ се изскачащ прозорец предпазва потребителя и от натискане на разни други обекти на екрана.

Пример: Динамично търсене

Много известен е примерът за търсене-докато-пишеш. В този пример ние ще покажем как може да се определя въвеждане на текст `TextInput` и на данни в търсенето, което се показва в таблица, докато потребителят пише.

Снимка:



Действието, определено в този случай, е “change”:

```
<TextInput change="dispatchEvent(form1,,action33)"/>
```

В този пример действието “action33” ще бъде повикано при всяка промяна в TextInput. Това действие ще изпълнява търсенето и ще връща отговор (само нови данни), като резултатът е показан в списъка с компоненти.

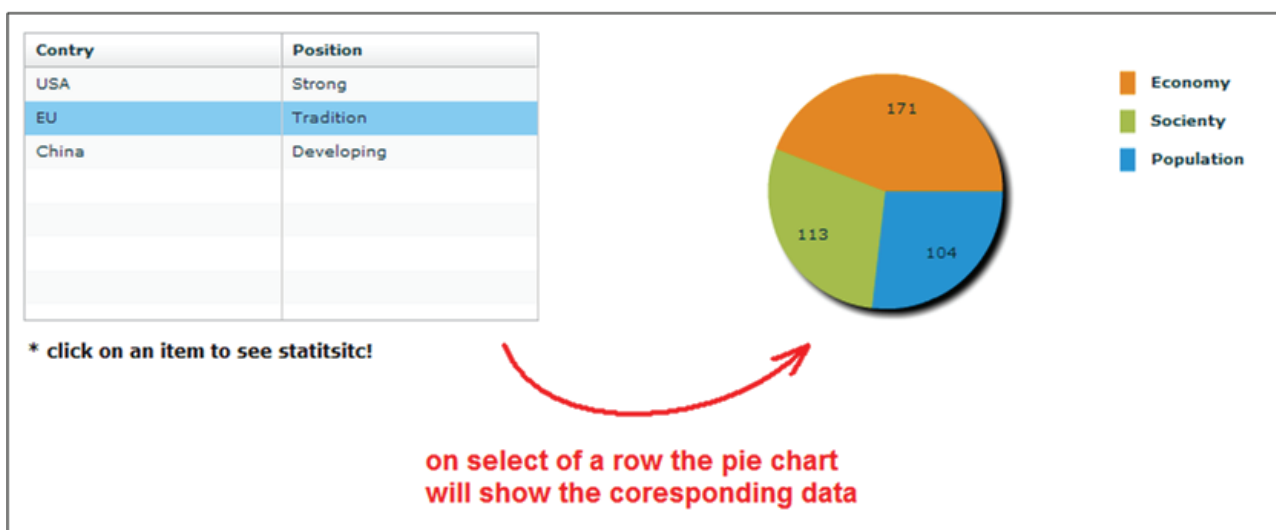
Пример: Таблица DataGrid във форма

Поставяме <DataGrid> във форма. Избраният ред на <DataGrid> ще бъде изпратен до действието. Данните в <DataGrid> се зареждат от XML компонент, който е част от <templatedata>. Не всички данни в този XML са видими за потребителя. Например в XML компонент може да има свойствено ID, което не е показано в <DataGrid>. Целият XML компонент е изпратен към действието и заявката може да се прочете. Това е много полезно, така че действието може да получи ID на избрания ред.

Пример: Натискане на обект ItemClick върху таблица DataGrid

За пример ние взимаме таблицата и кръговата диаграма. Когато потребителят избере ред от таблицата, кръговата диаграма следва да покаже графика, базирана на данните от избрания ред.

Снимка:





В този случай събитието “ItemClick” е определено. Ние можем да използваме свойствата “click” или “ItemClick”. Всъщност всяко натискане в таблицата, включително заглавията на колоните, ще извика действие. Във втория случай при натискане на реда ще се повика действие. Втората опция е тази, от която имаме нужда.

Пример:

```
<Box id="form1">  
  <DataGrid ItemClick="dispatchEvent(form1,,showpiechart)">  
    <columns>  
      <DataGridColumn headerText="Contry" dataField="name"/>  
    </columns>  
  </DataGrid>  
</Box>
```

В горния пример функцията, която определя действието има за първи параметър “form1” **dispatchEvent(form1,,showpiechart’)**.

Този параметър показва id-то на интерактивния контейнер, където се намира таблицата. В горния пример това е **<Box id="form1">**. Данните от всички интерактивни компоненти в тази кутия ще бъдат изпратени със заявка до действието. В този пример избраният ред от таблицата ще бъде изпратен до сървъра и сървърът ще върне данните за диаграмата. Със свързването на данните, данните от **<templatedata>** ще бъдат обновени директно върху кръговата диаграма.



СИСТЕМАТИЗИРАНЕ НА ВЪЗМОЖНИТЕ ТЕХНОЛОГИИ, ИЗПОЛЗВАНИ ОТ КЛИЕНТА И ПОДРОБНОТО ИМ ОПИСАНИЕ ЗА ПРИОРИТИЗИРАНЕ НА КОНКРЕТНИЯ ПРОБЛЕМ ИЛИ ВЪЗЕЛ ОТ ДЕЙНОСТТА

Следващите диаграми показват всички поддържани събития за всеки компонент

Component/Event	click	doubleClick	change	onChange	selectedItem	selectedIndex	itemClick	mouseover	mouseout	rollOver	rollOut	tooltipShow	tooltipHide	mouseUp	mouseDown	focusIn	focusOut	show	hide	scroll
Accordion	x	x			x		x	x	x	x	x	x	x	x	x	x	x	x	x	
AdvancedDataGrid	x	x		x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
ApplicationControlBar	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
Box	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
Button	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
Canvas	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
CheckBox	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	
ColorPicker	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	
Column Chart	x	x		x			x	x	x	x	x	x	x	x	x	x	x	x	x	
ComboBox	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
ControlBar	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
DataGrid	x	x		x		x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
DateChooser	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	
DateField	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	
Form	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
Grid	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
HBox	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
HDivideBox	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	
HLOC Chart	x	x					x	x	x	x	x	x	x	x	x	x	x	x	x	



Component/Event	click	doubleClick	change	onChange	selectedItem	selectedIndex	itemClick	mouseover	mouseout	rollOver	rollOut	tooltipShow	tooltipHide	mouseUp	mouseDown	focusIn	focusOut	show	hide	scroll
HRule	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
HSlider	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	x
Image	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
Label	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
Line Chart	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
LinkBar	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
LinkButton	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
List	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
MenuBar	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
NumericStepper	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x	x	x
Panel	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
Pie Chart	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
Plot Chart	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
RadioButton	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x		
RadioButtonGroup	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
Resizable Window	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
RichTextEditor	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x		
Spacer	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
TabBar	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
TabNavigator	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
Text	x	x					x	x	x	x	x	x	x	x	x	x	x	x		

Component/Event	click	doubleClick	change	onChange	selectedItem	selectedIndex	itemClick	mouseover	mouseout	rollOver	rollOut	tooltipShow	tooltipHide	mouseUp	mouseDown	focusIn	focusOut	show	hide	scroll
TextArea	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x		
TextInput	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x		
Tile	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
TitleWindow	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
Tree	x	x	x	x		x	x	x	x	x	x	x	x	x	x	x	x	x		
VBox	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
VDivideBox	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
VRule	x	x					x	x	x	x	x	x	x	x	x	x	x	x		
VSlider	x	x	x	x			x	x	x	x	x	x	x	x	x	x	x	x		x

ОРГАНИЗАЦИЯ НА ДАННИТЕ И ИНФОРМАЦИОННИТЕ ПРОЦЕСИ ПО ПРЕДЛАГАНЕ НА ОНЛАЙН ПОДДРЪЖКА ОТ НАЧАЛО ДО КРАЙ

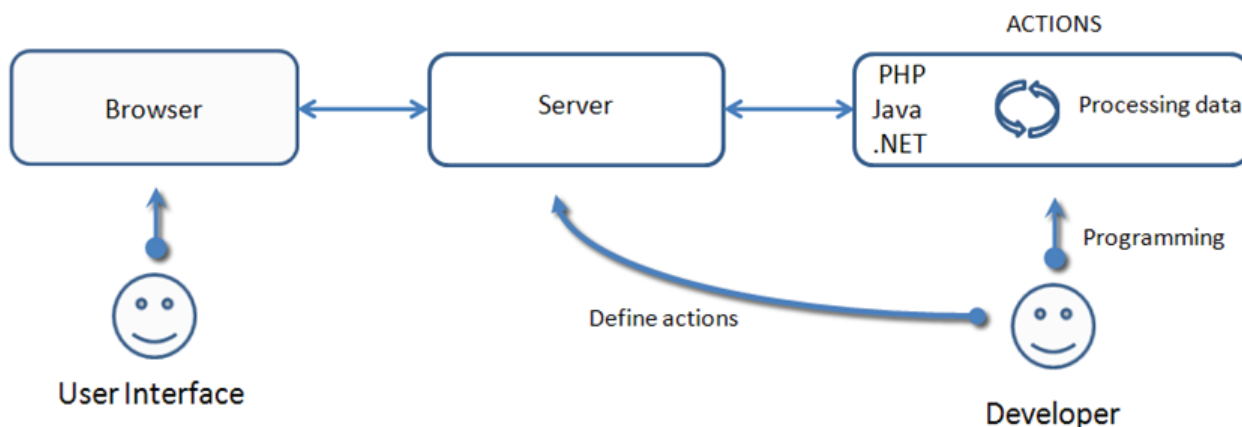
Обработка на форми (Действия за заявки и отговори)

В предишните глави беше описано дефинирането на събития и форми. В тази глава ще бъде обяснено точно какво се случва, когато действието е повикано и как данните, въведени от потребителя, могат да бъдат обработени.

Повикване на действие (заявка/отговор)

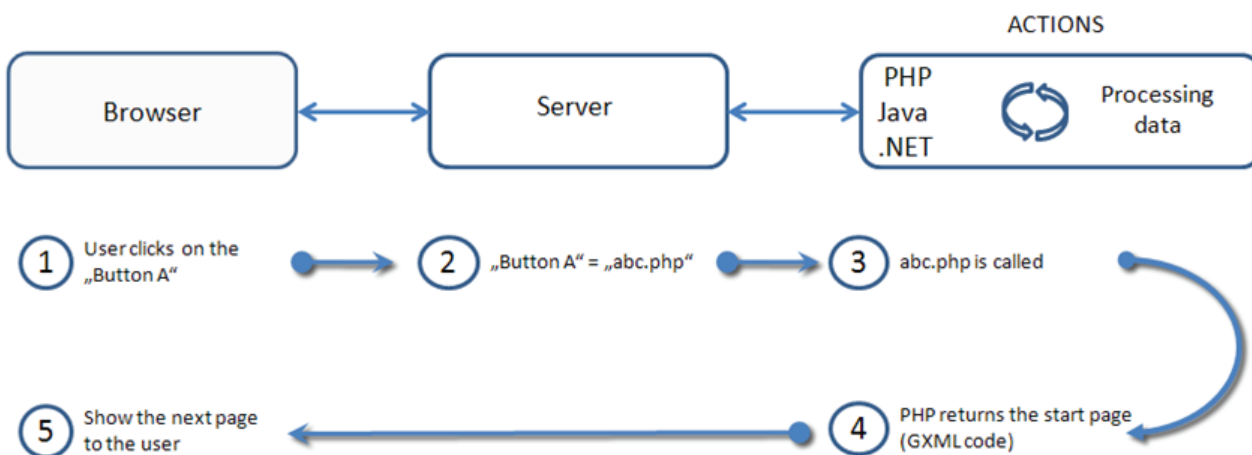
Действията са осъществени в програмния код на всяко DRIVEAPP приложение. Повикването на действие се прави от DRIVEAPP сървър. Заявката за действие е всъщност POST заявка с параметри.

Overview of components



Ето пример какво се случва, когато се натисне бутон:

User action => button click => show the next page



DRIVEAPP сървър работи като диспечер между потребителския интерфейс и действията.

Действието може да бъде програмирано във всяка интернет технология или език. Единственото условие е то да може да обработва HTTP/POST заявки. Това може да бъде PHP, Java, .NET, Ruby on Rails, Perl, CGI/C или всеки друг език зад уеб сървър.

В това описание ние няма да влизаме в детайли как да се изгради уеб страница, която може да обработва POST заявка. Това е специфично за всеки език. Но най-важното е, че това не е по-различно от обработване на POST заявка за уеб страница, когато е въведена HTML форма.

По същия начин, по който уеб страницата генерира HTML код, действието генерира DRIVEAPP код.

Един прост пример за PHP страница, която може да бъде използвана като действие.

PHP пример:

```
<?PHP
echo „<DRIVEAPPXML>“;
echo „ <template>“;
echo „ <Button>“;
echo „ </template>“;
echo „</DRIVEAPPXML>“;
?>
```



Фактът, че функциите на действието са по същия начин както всяка една динамична HTML страница, позволява да се използва персонална предпочитана технология или език, за да се програмират действията.

Изпращане на форма данни до действието

Когато действието е повикано, специални параметри преминават в POST заявка, също както се въвежда форма в HTML.

Системни параметри (включени във всяка DRIVEAPP заявка):

Параметър	Описание
action	Името на действието, което е извикано и дефинирано във функцията <code>dispatchEvent()</code>
event_id	ID-то на натиснатия компонент. Например: <code><Button id="abc">...</Button></code>

Със системните параметри, името на действието и id-то на събитието, DRIVEAPP ще изпрати също и интерактивните компоненти, ако в `dispatchEvent()` function е специфицирано валидно form-ID .

Параметри на формата, изпратени със заявка:

Параметър	Описание
<code>< form-component-id></code>	За всеки компонент във форма, с негово специфично id.

Веднъж повикано действието, данните се предават към него като POST променливи. Във всяка технология данните от POST заявката са достъпни през специална променлива или обект:

PHP	\$_REQUEST — HTTP Request variables
JavaServlet	the HttpServletRequest object
ASP.NET	Request.Form("componentId")

Бележка: Обработката на форми данни в DRIVEAPP е същата като обработката на форма в HTML.

Действието следва най-накрая да напише извод към отговора. Този извод трябва да бъде валидна DRIVEAPP структура на страница.

Качване на файлове

В DRIVEAPP можете да се качват файлове чрез елемент `<FileUpload>`

```
<FileUpload labelbutton="Select File"/>
```

Качване на файлове

Ако се използва FileUpload компонент във формата, DRIVEAPP ще включи файла в POST заявката по същия начин, както се получава файл от HTML форма.

PHP пример:

< Това е примерен код, за по-детайлна информация може да разгледате PHP документация>

```
<?php
if ($_FILES["file"]["error"] > 0) {
    echo „Error:„ . $_FILES["filedata"]["error"] . „<br />“;
} else {
    echo „Upload:„ . $_FILES["filedata"]["name"] . „<br />“;
    echo „Type:„ . $_FILES["filedata"]["type"] . „<br />“;
    echo „Size:„ . ($_FILES["filedata"]["size"] / 1024) . „ Kb<br />“;
    echo „Stored in:„ . $_FILES["filedata"]["tmp_name"];
```



```
//save the file to your target directory and name
move_uploaded_file($_FILES[„Filedata“][„tmp_name“],„<your new file name>“);
}
?>
```

Java/JSP пример:

< Това е примерен код, за по-детайлна информация може да разгледате JAVA документация >

```
<%
String savePath="";
String parentPath=request.getParameter(„Filedata“);

try {

    System.out.println( „CONTENT-LENGHT“ + request.getContentLength());
    System.out.println(„session id = „ + session.getId());

    DiskFileUpload fu = new DiskFileUpload();
    // maximum size before a FileUploadException will be thrown
    fu.setSizeMax(5000000);
    // maximum size that will be stored in memory
    fu.setSizeThreshold(4096);
    // the location for saving data that is larger than getSizeThreshold()
    fu.setRepositoryPath(„/tmp“);

    List fileItems = fu.parseRequest(request);
    Iterator item = fileItems.iterator();
    while(item.hasNext())
    {
        FileItem fi = (FileItem) item.next();
        if(fi.isFormField())
        {
            continue;
        }

        String[] parts = fi.getName().split(„/“);
        String fileName = parts[parts.length-1];

        if(fileName != null && !fileName.equals(„“))
        {
            out.println(„File name:“ + fileName + „, File size:“ + fi.getSize());
        }
    }
}
```



```

        if(! savePath.equals(""))
        {
            File f = new File(savePath + fileName);
            System.out.println("f = " + f.getAbsolutePath());
            fi.write(f);
        }
    }
}
}
catch (Exception ex) {
    // ex.printStackTrace();
}
%>

```

.NET пример:

< Това е примерен код, за по-детайлна информация може да разгледате .NET документация >

```

if( ( File1.PostedFile != null ) && ( File1.PostedFile.ContentLength > 0 ) )
{
    string fn = System.IO.Path.GetFileName(File1.PostedFile.FileName);
    string SaveLocation = Server.MapPath("Data") + "\\ " + fn;
    try
    {
        File1.PostedFile.SaveAs(SaveLocation);
        Response.Write("The file has been uploaded.");
    }
    catch ( Exception ex )
    {
        Response.Write("Error: " + ex.Message);
        //Note: Exception.Message returns a detailed message that describes the current
exception.
        //For security reasons, we do not recommend that you return Exception.Message to
end users in
        //production environments. It would be better to put a generic error message.
    }
}
else
{
    Response.Write("Please select a file to upload.");
}

```



Сесия

Използването на сесия става по същия начин както в HTML приложение. Различно е за всяка технология, но по същия начин, както се започва и използва сесията във уеб приложението, така се случва и в DRIVEAPP.

PHP пример:

```
<?PHP  
Session_start();  
$_SESSION[anyvar] = 777;  
?>
```

JAVA пример:

```
...  
HttpSession session = request.getSession(true);  
...  
session.setAttribute("name","data");  
String value = session.getAttribute("name").toString();  
...
```

FastEdit = История на действието

FastEdit е мониторинг инструмент, достъпен за всяко DRIVEAPP приложение – за да покаже комуникацията между DRIVEAPP и действията. Точните параметри от заявките и съответните отговори могат да бъдат показани за последните извършени действия.

FastEdit е подобен на функцията “ViewSource” в браузър, но той показва не само последната заявка, а също и историята на заявката.

Данни в POST заявка

В post заявката всички данни са URL-кодирани (Percent-encoding). Следователно, следва да се използва URL-декодираща функция, за да се трансформират данните на параметрите.



Бележка: URL Кодирането, още известно като Percent-encoding, е механизъм за кодиране на специални символи в Uniform Resource Identifier (URI) при определени обстоятелства. Това URL позволява само ASCII символи a-z и A-Z, ако искаме да използваме например `space`, то ще бъде кодирано като `%20`. Ако искаме да го обърнем обратно в `space`, използваме специфичните за избрания език url-декодиращи функции.

PHP пример (url-декодиране):

```
<?PHP
$result = urldecode($_REQUEST[„parameter_name“]);
echo „<DRIVEAPPXML>“;
echo „ <template>“;
echo „ <Text text=“$result“>“;
echo „ </template>“;
echo „</DRIVEAPPXML>“;
?>
```

За всеки компонент във формата, ще се получи параметър в POST заявката.

Например за `TextInput` с име `“username”` в заявката ще бъде посочен параметър `“username=<data-entered-by-the-user>”`, който може да бъде взет от нея.

PHP пример:

```
<?PHP
$result = urldecode($_REQUEST[„username“]);
...
```

Ако има `DataGrid`, `ComboBox` или друг компонент, където потребителят може да направи избор/селекция, в POST заявката ще бъде поставен параметър с името на интерактивен компонент и суфикс `“selectedItem”`.

Пример: `orders_selectedItem = <node id=“9” name=“Order X”/>`

В този случай името на параметъра е `<form-component-ID> + “_” + “selectedItem”`. В този случай избраният ред от `DataGrid` или `ComboBox` е представен от XML компонент в `<templatedata>` секция. Така може да се получи точният XML компонент: `<node id=“9” name=“Order X”/>`



В действието следва да се вземе този XML и да се получат данните, които са нужни. Ако има нужда от ID на избрания компонент, ще се получи параметърът **ID**.

PHP пример:

```
<?PHP
$xmlString = urldecode($_REQUEST[„parameter_name“]);
$xmlObject = new SimpleXMLElement($xmlString);
$selected_id = $xmlObject->id;
$selected_name = $xmlObject->name;
...
```



ВИЗУАЛИЗИРАНЕ НА ВЗАИМОДЕЙСТВИЕТО НА ВСИЧКИ ЕЛЕМЕНТИ И ПРОЦЕСИ ПРИ ПРЕДЛАГАНЕ НА УСЛУГАТА. КОМПОНЕНТИ

Всеки компонент в MXML синтаксис има свойства:

```
<COMPONENT property1="value1" property2="value2" ... >
```

MXML изисква строга XML структура, така че всеки компонент следва да се затваря.

Пример за компонент, затворен на същото поле с `</>`:

```
<COMPONENT property1="value1" property2="value2" />
```

Пример за компонент, затворен със затварящ таг:

```
<COMPONENT property1="value1" property2="value2">  
</COMPONENT>
```

Списък на компоненти и свойства в *DRIVEAPP*

Форми

Form-components	Description
AdvancedDataGrid	The AdvancedDataGrid control extends the capabilities of the standard DataGrid control with: sorting by multiple columns displaying hierarchical and grouped data
Button	Common rectangular button. Label or icon is supported.
CheckBox	Commonly used graphical control that can contain a check mark.
ColorPicker	The ColorPicker control lets users select a color from a drop-down swatch panel (palette).
ComboBox	j
DataGrid	Multi column display of data.



DateChooser	Calendar from which the user can choose a date.
DateField	The DateField control is a text field that shows the date with a calendar icon on its right side.
HSlider/VSlider	You can use the slider controls to select a value by moving a slider thumb between the end points of the slider track.
LinkButton	The LinkButton control is a borderless Button control whose contents are highlighted when a user moves the mouse over it.
List	The List control displays a vertical list of items. The user can select single or multiple values.
NumericStepper	The NumericStepper control lets the user select a number from an ordered set.
RadioButton	The RadioButton control lets the user make a single choice within a set of values.
RadioButton-Group	By using the property groupName="cardtype" you can create a group of RadioButtons. Within this group only one value can be selected.
RichTextEditor	The RichTextEditor control lets users enter, edit, and format text.
TextArea	The TextArea control is a multiline text field.
TextInput	The TextInput control is a single-line text field.
Tree	The Tree control allows you to show hierarchical data arranged as an expandable tree.

Оформление

ApplicationControlBar	You use the ApplicationControlBar container to hold components that provide access to application navigation elements and commands.
Box	A Box container lays out its children in a single vertical column or a single horizontal row. The direction property determines whether to use vertical (default) or horizontal layout.
Canvas	A Canvas layout container defines a rectangular region in which you place child containers and controls.



ControlBar	You use the ControlBar container with a Panel or TitleWindow container to hold components that can be shared by the other children in the Panel or TitleWindow container.
Form	Collation of form elements. This task can be performed with any other container element.
Grid	A Grid container lets you arrange children as rows and columns of cells, similar to an HTML table.
HBox	The HBox container lays out its children in a single horizontal row.
HDivideBox	The HDividedBox container lays out its children horizontally in the same way as HBox container, but it inserts a draggable divider in the gap between each child.
HRule	The HRule (Horizontal Rule) control creates a single horizontal line and the VRule (Vertical Rule) control creates a single vertical line. You typically use these controls to create dividing lines within a container.
Panel	A Panel container consists of a title bar, a caption, a border, and a content area for its children.
ResizableWindow	The ResizableWindow container shows the content in a resizable pop-up window.
Spacer	The Spacer control helps you lay out children within a parent container. It does not draw anything, it only allocates empty space.
Tile	A Tile layout container lays out its children in one or more vertical columns or horizontal rows, starting new rows or columns as necessary. The direction property determines the layout.
TitleWindow	A TitleWindow layout container contains a title bar, a caption, a border, and a content area for its child.
VBox	The VBox container lays out its children in a single vertical column.



VDivideBox	The VDividedBox container lays out its children vertically in the same way as the VBox container, but it inserts a draggable divider in the gap between each child.
VRule	The HRule (Horizontal Rule) control creates a single horizontal line and the VRule (Vertical Rule) control creates a single vertical line. You typically use these controls to create dividing lines within a container.

Навигация

Accordion	An Accordion navigator container has a collection of child containers, but only one of them at a time is visible.
MenuBar	A MenuBar control displays the top level of a tree-structured menu as a horizontal bar of menu items.
LinkBar	A LinkBar control is a vertical row of LinkButton controls that function as Tabs.
TabBar	A TabBar control defines a horizontal or vertical row of tabs.
TabNavigator	The TabNavigator control is used to navigate between Tabs.

Диаграми

Column Chart	The ColumnChart control represents data as a series of vertical columns.
HLOC Chart	The HLOCChart (HighLowOpenClose) control represents financial data as a series of lines representing the high, low, opening, and closing values of a data series.
Line Chart	The LineChart control represents a data series as points connected by a continuous line.
Pie Chart	The PieChart control represents a data series as a standard pie chart.
Plot Chart	You use the PlotChart control to represent data in Cartesian coordinates where each data point has one value that determines its position along the x-axis, and one value that determines its position along the y-axis.

GTimer компонент

Компонентът **<GTimer>** в DRIVEAPP ще изпрати автоматично нормална заявка до сървъра за обновяване на данните на екрана.

Пример:

```
<GTimer id="mytimer" seconds="5000" pause="false" action="dispatchEvent(<FORM-ID>,<ACTION-NAME>)" />
```

Тази команда ще задвижи таймер, параметърът **seconds** ще дефинира през какви периоди заявките да бъдат изпратени към сървъра, за да се обновяват данните. Параметърът е в милисекунди. В горния пример стойността 5000 е равна на 5 секунди. Това означава, че на всеки пет секунди данните се обновяват от сървъра.

Бележка: GTimer е като всеки друг компонент и може да бъде добавен към текущия екран. Може да се обновяват параметрите на таймера като секунди и пауза, таймерът може да се премахва с remove команда; таймерът ще бъде премахнат и с командата remove_all

Включване на таймера

Таймерът може да се включи със следната команда:

```
<GTimer id="mytimer" seconds="5000" pause="false" action="dispatchEvent(<FORM-ID>,<ACTION-NAME>)" />
```

Това ще включи таймера със id „mytimer“, което ще изпрати заявка до **<ACTION-NAME>** и ще прикачи данните от **<FORM-ID>** на всеки пет секунди.

Бележка: ID-то на таймера е много важно. Може да се използва, за да обновява свойствата на таймера като секунди или пауза и също за премахване.



Спиране на таймера

Има две опции за спиране на таймера:

a) Премахване на таймера с команда **<remove>**

Пример:

```
<DRIVEAPPXML>  
  <commands>  
    <remove target="GTIMER-ID ,"/>  
  </commands>  
</DRIVEAPPXML>
```

b) Настройване на свойството **pause** към true. Може да се използва **<templatedata>** и **<Binding>**, за да се настрои свойството за пауза на таймера към true или false.

Пример: Да се спре таймера, като се настрои свойството пауза към **true**:

```
<DRIVEAPPXML>  
  <template>  
    <GTimer id="mytimer" seconds="5000" pause="true" action="dispatchEvent(dynamic,  
,timeraction')"/>  
    <Binding source="templatedata.timerstatus" destination="mytimer.pause"/>  
  </template>  
  <templatedata>  
    <!-- setting the pause to true will pause/stop the timer -->  
    <timerstatus>true</timerstatus>  
  </templatedata>  
</DRIVEAPPXML>
```



Поддръжка на CSS

Може да се спести много работа със стилизиране на приложението с CSS (Cascading Style Sheets)!

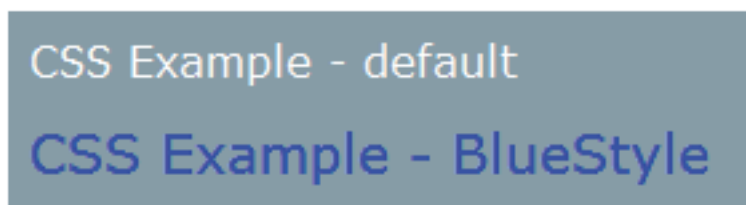
Тук се описва как да прилагаме CSS към DRIVEAPP приложение. В DRIVEAPP стилове (CSS) са поддържани за повечето от компонентите.

Командите за стил могат да бъдат записани в `<template>` тага.

Пример:

```
<DRIVEAPPXML>
  <template>
    <Style>
      Label{
        fontSize:"16";
        color:white;
      }
      .BlueStyle{
        fontSize:"20";
        color:blue;
      }
    </Style>
    <VBox>
      <Label text="CSS Example - default" />
      <Label text="CSS Example - BlueStyle" styleName="BlueStyle"/>
    </VBox>
  </template>
</DRIVEAPPXML>
```

Снимка:





Използването на CSS в Джјенерик Фрейм Онлайн Студио е също толкова лесно, както използването на CSS в HTML страница. В CSS мјоже да се включат общи стилове, които се използват от множество от компоненти. Например, може да се определят общи свойства за всички бутони:

Пример:

```
<Style>  
  Button {  
    fontSize:"16";  
    color:white;  
  }  
</Style>
```

Може също да се използва идентификационни имена за стилове:

```
<Style>  
  .myName {  
    fontSize:"16";  
    color:white;  
  }  
</Style>
```

Това име (.myName) следва да започва с точка. То няма да бъде автоматично приложено към всеки компонент. Следва самостоятелно да определим компонента, където трябва да бъде приложено.

```
<Button text="Buttn One" styleName="myName"/>
```

Стиловете, определени в **<Style>** секцията, могат да бъдат пренаписани чрез подробно присвояване на свойства на компонента. Например в **<Style>** секцията за бутоните, размерът по подразбиране може да бъде настроен на 16, но при желание може да има един бутон с размер 12, ако се напише в кода на бутоните.

```
<Button text="Buttn One" fontSize="12"/>
```

Бележка: Препоръчително е да се избягва зареждането на Стили на всеки екран. Следва да се зареди CSS-ът в началото на приложението при първия екран. Така ще се избегне ненужно показване на елементите на екрана отново и отново.