Storyteller - Developer Manual

Denna manual är en guide för att komma igång med vidareutveckling av applikationen Storyteller. Applikation är byggd på enheter som kör Android som operativsystem. Manualen går igenom applikationen och dess funktionalitet samt hur vidareutveckling ska ske med avseende på design och kodning. Utvecklingsverktyget som rekommenderas är Android Studio eftersom detta verktyg bidrar med funktionalitet som gör att mindre manuellt arbete blir nödvändigt.

Get the source from github

Projektet finns tillgängligt på github.
Git clone från https://github.com/larssonwilly/Story-app-2

Dependencies

För att kunna kompilera och köra applikationen behöver du förutom källkoden ha:

- Android device (Android 4 eller högre)
- Android SDK
- Virtual Android Device
- Java 7 SE development environment
- Parse-1.9.1.jar

Projektets build.gradle ser ut enligt Figur 1:

Figur 1: Projektets build.gradle som visar dess dependencies

Android SDKs

Minimumkrav för version på SDK är 14 och target SDK är 22. Vår build.gradle ser ut enligt Figur 2:

Figur 2: Projektets build.gradle som visar versionen på SDK som en användare behöver

Building and installing

Appen är byggd med användning av gradle build. För att bygga projektet i Android Studio, klicka *build* och välj *make project*.

Application description

Applikationen storyteller har funktionaliteten att användare kan bidra till att skriva en historia med valfritt ämne. I applikationen är det möjligt att se de sista 100 tecknen på historien och användare måste skriva minst 15 tecken för att kunna bygga vidare på historien. Användaren blir ansluten till en historia tills denna är färdigskriven och då visas hela historien.

Java classes

Förklaring över vad de olika klasserna i applikationen fyller för funktion.

- Activites
    - AuthenticateActivity
        - Inloggnings/registreringsskärmen. Beroende på vilken knapp som användes för att ta sig till denna vy från startskärmen fungerar denna vy antingen som registrering eller inloggning.
    - MainActivity
        - Startskärmen för applikationen. Möjlighet till att välja logga in eller registrera sig, och då skickas användaren till AuthenticateActivity. Är användaren inloggad skickas denne direkt till AuthenticateActivity.
    - StoryModePresenter
        - Hanterar all logik för att kunna skapa och fortsätta stories.
    - StoryModeViewActivity
        - Ger användaren en möjlighet att ge sitt textbidrag till en story samt se de sista 40 tecknen på storyn som användaren är kopplad till. Här går det också att skapa nya stories, logga ut och se färdiga stories genom att välja detta i menyn.
    - StoryShowcaseActivity
        - Användaren kan se alla färdiga stories och vilka denna deltagit i.
    - AfterPostActivity
        - Aktiviteten man kommer till efter att man skrivit en post. Här finns valet att skapa en ny story eller att se alla stories som är färdiga.
    - EasterEggActivity
        - Ett roligt tillägg i appen, skriver man orden “fest” och “Hammaro” som input kommer denna aktiviteten upp som spelar en låt och visar en bild
- Helpers
    - StoryShowcaseAdapter
        - Skräddarsydd adapter för applikationens syften som utökar funktionaliteten i “ArrayAdapter”. Används för att kunna mata in data till en ListView i StoryShowCaseActivity.
    - StorylistHandler
        - Hanterar alla listor med posts och stories, och strukturerade metoder för att tillåta aktiviteterna och presentatörerna att enkelt hämta datan.

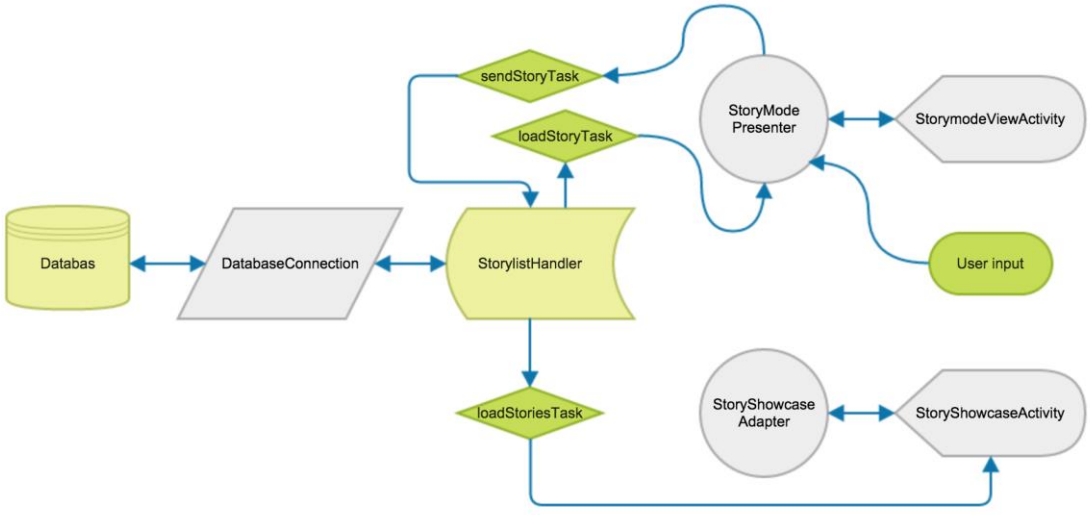
    - DatabaseConnetion
        - Ansluter direkt till Parse och skickar sedan vidare data till StorylistHandlern
    - TextHelper
        - Hanterar formatering av text åt StoryModePresenter
- Utils
    - ColorPalette
        - Enkel util för att ha applikationens färgpalett enkelt åtkomlig.

Architecture

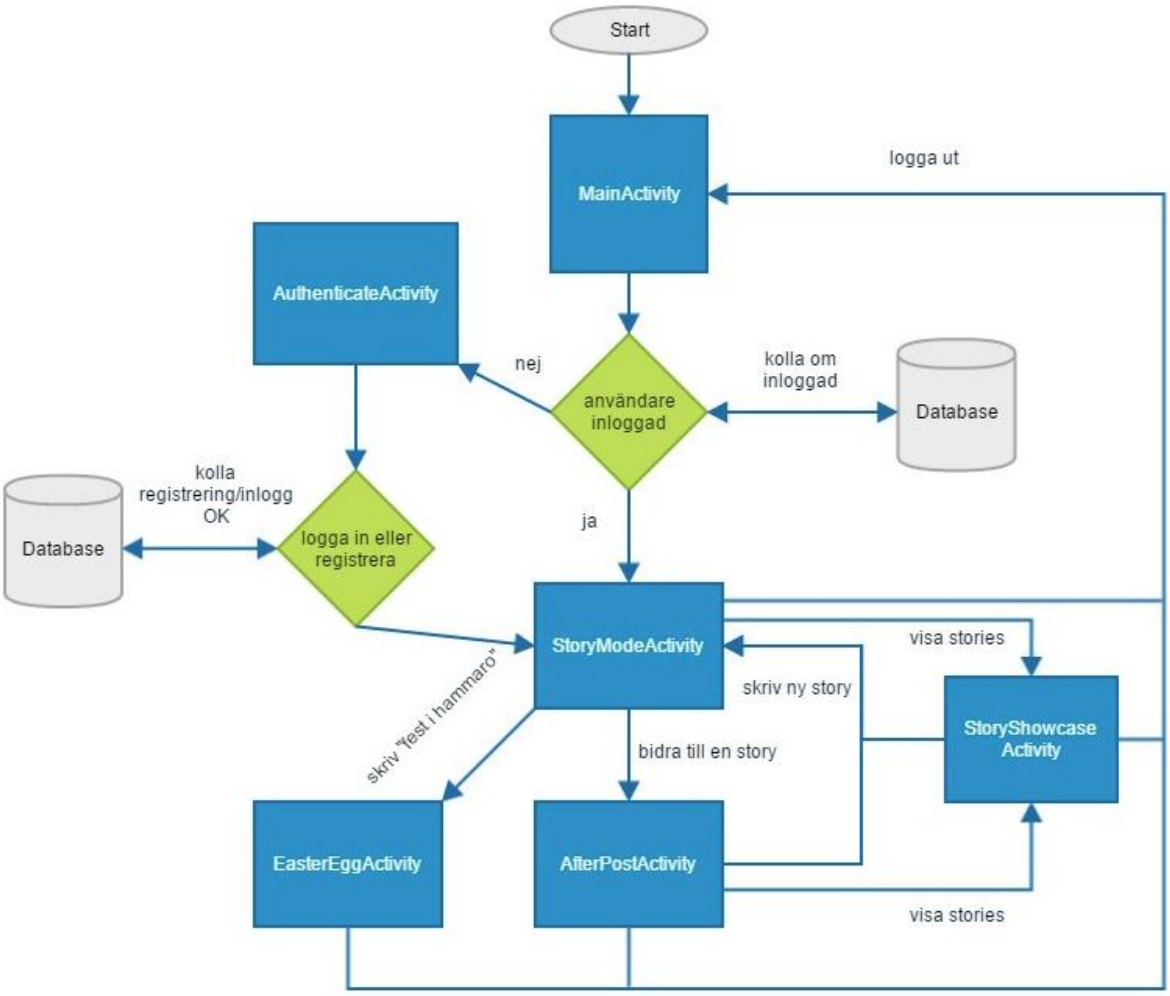
Applikationen är huvudsakligen byggd på en Model-View-Presenter-struktur för att enkelt kunna separera logik, grafik, och data. Mindre aktiviteter har dock fått hantera sin egen logik då presentatörer med endast en metod anses vara onödig separation av kod.

För större aktiviteter gäller det att logik hanteras av en presentatör, att grafiken hanteras av vyn, och att datahanteringen hanteras av en separat modell. Vyn ska endast känna till presentatören, presentatören ska endast känna till vyn, datahanteringsmodellen och eventuella hjälpklasser. Modellen ska i sin tur endast känna till presentatören och databasinterfacet. Databasen ska endast känna till datahanteringsmodellen.

Data flow



Figur 3: Flödesschema över dataflöden

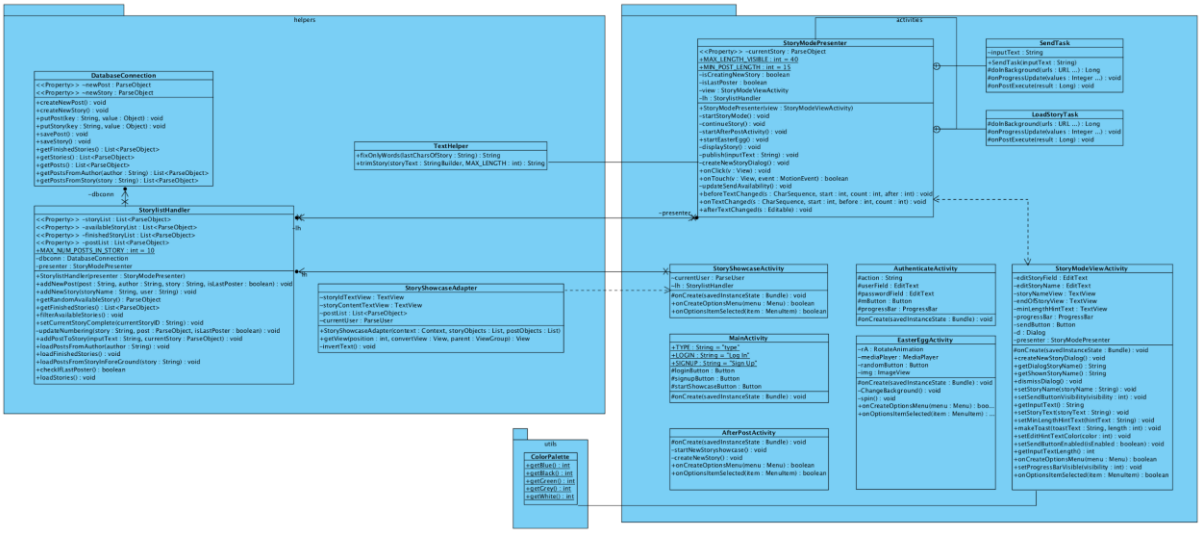


Figur 4: Visuell överblick över skärmvyerna

I Figur 4 visualiseras hur de olika skärmvyerna associerade med de olika klasserna hänger samman.

UML

I Figur 5 beskrivs applikationens klasser genom ett UML-diagram. Diagrammet genererades genom Visual Paradigm.



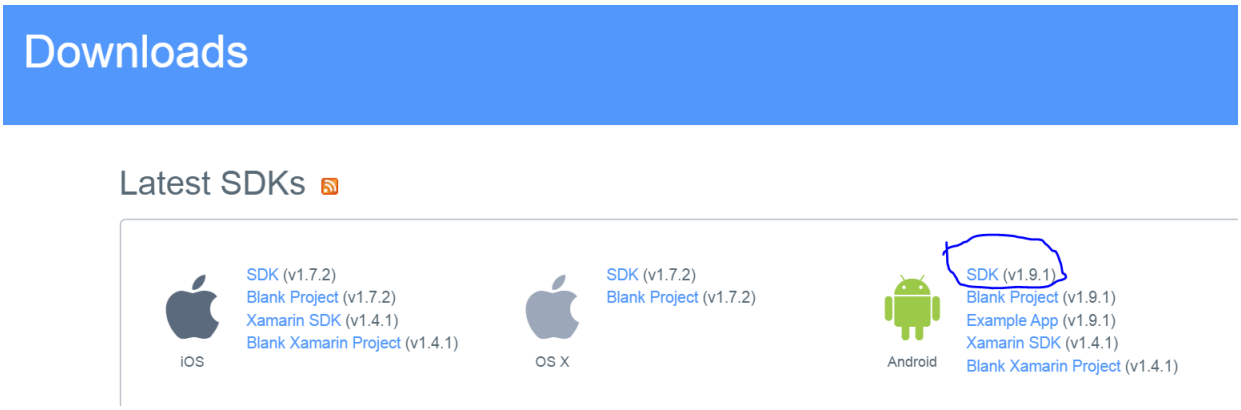
Figur 5: UML-diagram över applikationen

Vad är Parse och varför används det

Parse är en tjänst som används för att lagra data i molnet. Gruppen har valt Parse för att slippa serverunderhåll och komplicerad uppbyggnad. Med Parse kan vi lägga till push-meddelanden, datalagring med mer genom SDK:er från Parse. Vi har möjligheten att ansluta användare till applikationen via traditionell inloggning med bara några få rader kod och genom Parse har vi också guider att följa vilket hjälper oss att lösa olika problem som uppstår.

Hur Parse läggs till

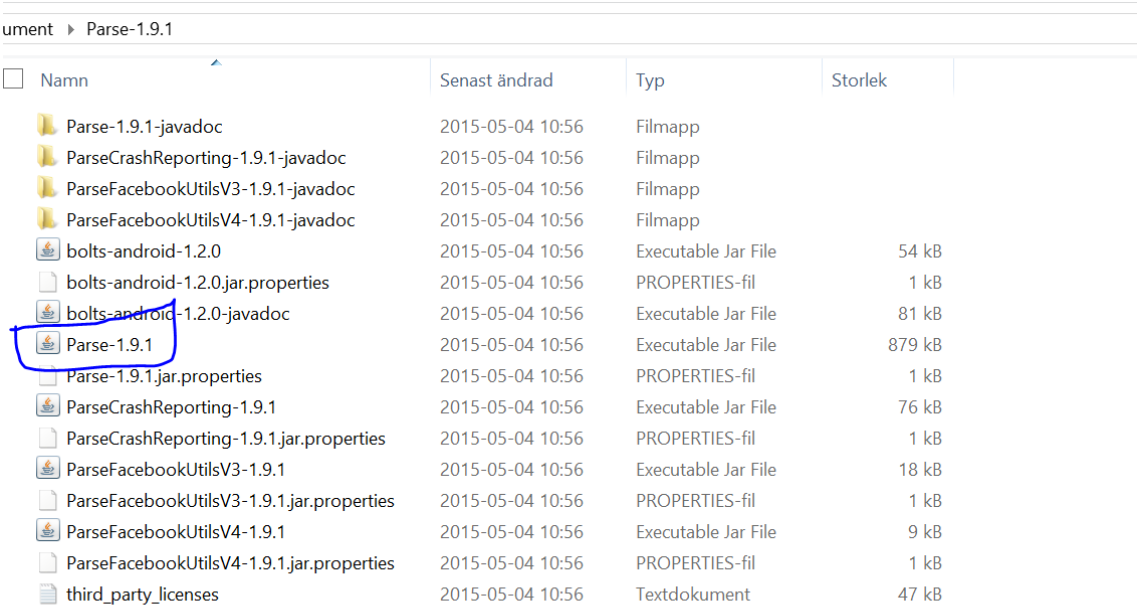
1: Ladda ner Parse SDK (v.1.9.1) från Parse hemsida: https://parse.com/docs/downloads (Figur 6)



Figur 6: Parse SDK (v.1.9.1) som finns på Parse hemsida är markerad

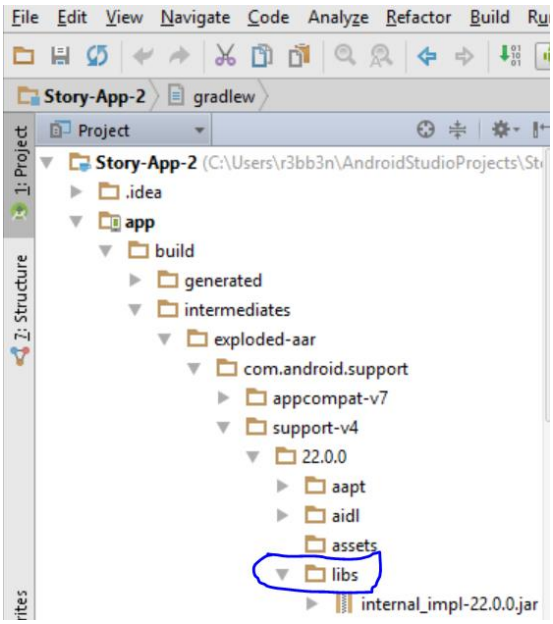
2: Extrahera filerna på datorn

3: Gå in i Parse-mappen och kopiera (ctrl+c) “Parse-1.9.1.jar” (Figur 7)



Figur 7: Den fil som ska kopieras är markerad

4. Gå in i Android Studio och leta upp mappen “libs”. Kopiera in “Parse-1.9.1.jar” i “libs”. Nu ska Parse fungera. En i gruppen behövde högerklicka filen och välja “add to library”, men de andra i gruppen har inte behövt göra detta för att få det att fungera. (Figur 8)



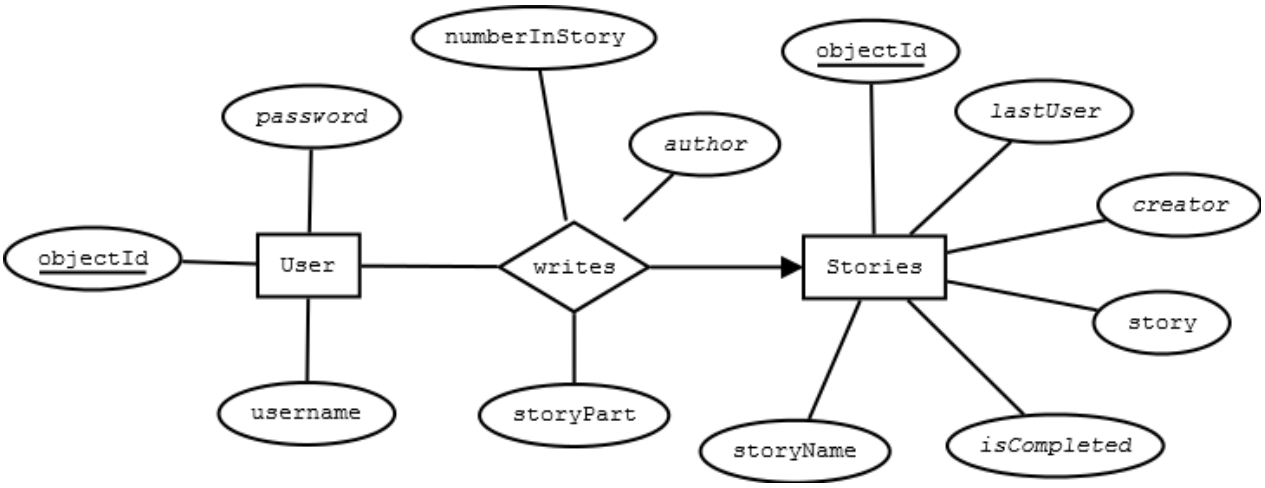
Figur 8: libs där filen ska klistras in är markerad

4.1. Om mappen “build” inte finns, gå in i filen “app.iml” och ta bort “<excludeFolder url="file://$MODULE_DIR$/build/intermediates/libs" />” raden. (Figur 9)

Figur 9: Den rad kod som ska raderas ur app.iml är markerad

ER-diagram för serverlösning

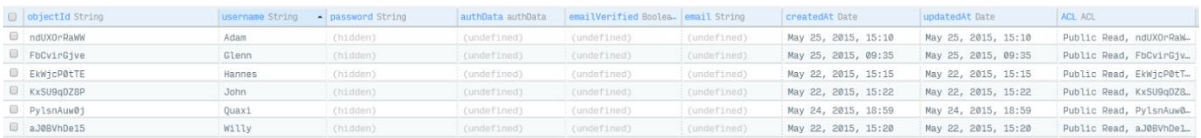
Bilden nedan visar ett ER-diagram över hur databasen i vår serverlösning är upplagd (Figur 10). Diagrammet innehåller entiteterna User och Stories, deras relation är Writes. User har attributet objectId som primärnyckel. Stories primärnyckel är också ett attribut som heter objectId. Relationen Writes har Users objectId och Stories objectId som referensnycklar.



Figur 10: ER-diagram över databasen i Parse

User table

I User lagras användarens användarnamn och lösenord (Figur 11). Varje användare har ett unikt id kopplat till sig. User-tabellen används av applikationen bland annat genom inloggning av användare och registrering.

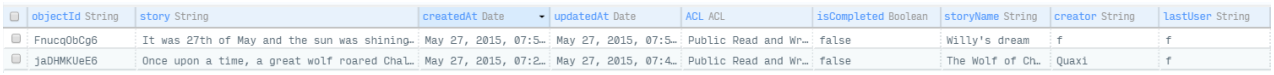


Figur 11: Tabellen som lagrar användare i Parse

Stories table

I Stories lagras berättelserna, om den är färdigställd och vilken användare som skapat den (Figur 12). Tabellen lagrar också vem som sist skrev i den och berättelsens titel. Till varje berättelse finns ett unikt id kopplat. Tabellen används av applikationen t.ex. för att visa

färdigskrivna berättelser, när en ny story skall skapas och när en berättelse får ett tillägg från en användare.



objectId String	story String	createdAt Date	updatedAt Date	ACL ACL	isCompleted Boolean	storyName String	creator String	lastUser String
FnucqObCg6	It was 27th of May and the sun was shining…	May 27, 2015, 07:5…	May 27, 2015, 07:5…	Public Read and Wr…	false	Willy's dream	f	f
jaDHMKUeE6	Once upon a time, a great wolf roared Chal…	May 27, 2015, 07:2…	May 27, 2015, 07:4…	Public Read and Wr…	false	The Wolf of Ch…	Quax1	f

Figur 12: Tabellen som lagrar berättelser i Parse

Writes relation

I Writes lagras de individuella bidragen till berättelserna (Figur 13). Tabellen innehåller vem som skrivit bidraget, vad bidraget är och vilket i ordningen i berättelsen det är. Till bidragen är nycklar från både User och Stories kopplade. Tabellen används av applikationen t.ex. när en användare skriver ett bidrag till en berättelse och detta skall läggas till.

objectId String	author String	inStory String	createdAt Date	storyPart String	numberInStory Numbe…	updatedAt Date	ACL ACL
pPHBJvmIQ4	f	FnucqObCg6	May 27, 2015, 07:53	It was 27th of May and the sun was shining bright like never before. Willy …	1	May 27, 2015, 07:53	Public Read and Writ…
0GMIwpdOMY	f	jaDHMKUeE6	May 27, 2015, 07:44	big Willy. Even though his amazing super powers, he never	1	May 27, 2015, 07:44	Public Read and Writ…
K9FDaHtxEy	Quax1	x4zWwrCh1P	May 27, 2015, 07:31	Fest i hammaro!	0	May 27, 2015, 07:31	Public Read and Writ…
i1RVkfd6r0	Quax1	jaDHMKUeE6	May 27, 2015, 07:26	Once upon a time, a great wolf roared Chalmers. He was a fiercesome beast w…	0	May 27, 2015, 07:26	Public Read and Writ…

Figur 13: Tabellen som lagrar de enskilda individernas bidrag till varje berättelse i Parse

Git-standarder

Modellen är baserad på Vincent Driessens "A Sucessful Git Branching Model" med små modifikationer för att bättre passa vår grupps egna projekt.

Huvudbranscherna:

Projektet utgår från två huvudbrancher med oändlig tidslinje:

- master
- develop

"master" representerar färdig produktionskod. Då applikationen under utvecklingstid är väldigt grundläggande representerar detta oftast den senast fungerande versionen.

"develop" representerar den senaste versionen av utvecklingskod, så kallade "nightly" builds. När develop når en stabil punkt och är redo för release mergas den in till master och bemärks med ett releasenummer. Varje gång saker mergas tillbaka till master betyder detta *per definition* att det är en ny produktionsrelease.

Stödjande brancher:

Tillsammans med huvudbrancherna använder modellen även en varietet av stödjande brancher för att underlätta parallell utveckling, enklare spåra nya utvecklingar, och att snabbt fixa produktionsproblem. Till skillnad från huvudbranscherna så har dessa brancher en begränsad livslängd, och tas bort efter dem är mergade in i någon huvudbransch.

Varje bransch har ett specifikt syfte, och är bunden till specifika regler kring vem som får pusha till vilken bransch. Tekniskt sett är inga branscher speciella, utan alla är vanliga Git-branscher. Vad som definierar dem är hur dem används.

Feature-branscher:

Kan skapa egen bransch från:
    develop
Måste mergas in i:
    develope
Namnkonvention:
    featureNamnPåDinFeature

Feature-branscher används för att utveckla ny funktionalitet till applikationen. En feature-bransch existerar så länge som funktionaliteten utvecklas. Vid uppnådd funktionalitet mergas branschen sedan ihop i develop, eller förkastas om funktionaliteten anses onödig eller bristfällig.