شرحنا ثلاث أنواع من الموديلي بيو كونترولر، بعدين عملوا فيرجن حديث منه للويب، وبعدها بسموها موديلي بيو كونترولر ويب، وبعدها انتقلنا لأحدث إشي اللي هو الـjs ،single-page-application، هدول بيستخدموا الـprocessing أو الـbusiness logic بصفحة الـclient-side أو بصفحة البراوزر، بيكون عندك server-side وclient-side، الـspring-boot، الـspring-java، هدول كلهم بيشتغلوا على الـstyle الثاني للموديلي بيو كونترولر، اللي اسمه موديلي بيو كونترولر ويب. وشفنا أنه الـeffort البرمجي اللي الـdeveloper بيعمله أصعب من التطبيق نفسه، وصمم للـweb-applications اللي فيها complexity عليه. هذا الحكي كله حكينا ويمكن شرحنا الـexample كامل، هذا الـexample بس عشان تعرف أنه الـeffort اللي بتحطه بالـapplication البسيطة في الموديلي بيو كونترولر، رح يكون غير مناسب أنك تستخدم له هاي الـapplications البسيطة، Shutterstock طبعاً حكينا الـmonolithic-application كل أجزاء التطبيق العملي أو كل الموديولز اللي بتبنيها، بنعملها aggregation تجميع أو بتنحط في بروسس واحدة على الـoperating-system طبعاً في الـrun-mode، يعني لما تشغل التطبيق العملي جميع أجزاء التطبيق اللي هما عبارة عن موديولز بيكونوا، كلها بنعملها encapsulation، يعني هاد التطبيق كامل بيشتغل في الـrun-time على بروسس واحدة في نظامك التشغيل. بمعنى آخر إنه هاد التطبيق فيه dependency عالي بين الموديولز اللي موجودة عنه، فيه عنا خيارين حكينا، وفيه خيار تاني الـhorizontal scaling إللي هي تزيد عدد الأجهزة اللي موجودة عنا. لكن بالنسبة لمشاكل المونولوتو تيك أبليكيشن، بيعاني من مشكلة من ناحية الـscalability، ليش؟ لأنه ببساطة هاد التطبيق اللي بيشتغلوا كل الأجزاء الموديولز الموجودة فيه في بروسس واحدة، متطرة أجيب جهاز جديد وأنقل كل الموديولز الموجودة بالتطبيق على الجهاز الجديد، زاد عدد المستخدمين عليه، ما بقدر أعمل scaling لموديول 6 في المونولوتو تيك أبليكيشن، لازم أعمل scaling لكل التطبيق العملي اللي موجود على جهاز آخر، وهذا الحكي مش efficient، ليه؟ لأنك إنت عم تستهلك resources زي سيرفرات، عشان بس تعمل scaling لسيرفرات معينة من هذا التطبيق الموجود، فهذا تعتبر من المشاكل الأساسية الموجودة في المونولوتو تيك أبليكيشن. اللي هي تحرير نسخ من هذا الابليكيشن بطيئة جداً، ليه؟ لأنهم التيم اللي بيشتغلوا على هذا الابليكيشن، ما عندهم يعني بمعنى آخر الـpermission، السبب إنه أي تغيير في أي موديول من هاي الموديولز، لازم يكون جميع الموديولز عارفين عنها. ببساطة اللي بيصير كالتالي، ممكن يكون عدة developer بيشتغلوا في الابليكيشن، وباقي الموديول جاهزين، لهاي الموديولز الجاهزة، لحد ما ينتهوا شميع الموديولز من الـdevelopment، وبعدين ينعمل testing لكل الموديولز مع بعضها، وبعدين ينعمل releases لهم مرة واحدة، كل الموديولز الموجودة في الـmonolithic application موجودة في بروسس واحدة، كـmicro service موجود على بروسس، هذا الموديولز موجود على بروسس، وكل-micro، زي ما حكينا من الموديولز، كل شق منهم يعتبر-micro service بحد ذاتها، يعني هم بيحكوا مع بعض، لكن هم بشكل عام self-contained. ليه؟ لأنه ببساطة، اللي فيها module رقم 5، أو عملية installation لهاي الـservice لحالها، أو يعني ضغط من قبل اليوزر، لأنه independent عن الـservices الأخرى، وتنقله على سيرفر آخر، مش بالضرورة تعمل عملية installation لكل الـapplication، مقارنة بال-monolithic أسهل، ليه؟ لأنه ببساطة كل process، كل process بتشتغل بشكل منفصل، طبعاً، فمباشرة الـdeveloper بيقدر يعملها installation، ويعمل release من هاي الـmicro service اللي موجودة، على الـproduction side، يعني على سيرفرات رئيسية وتكون شغالة، أو كل الـmicro services تنتظر بعضها. مقارنة بال-monolithic، إنه ممكن يتبرمج كل-micro service على تكنولوجيا معينة، وأبرمجها على جزء منها على الـangular JS، فممكن تدعم multi-technologies فيهم. والشغلة كمان المهمة إنه الفيليارز اللي بيصير في التطبيق العملي بارشالي، لأنهم أصلاً هدول الـservices كل واحدة كأنها application بحد ذاتها، فهذا الإشي جاي مميز فعلياً بال-micro service architecture. بيطلبوا requests من بعض، فممكن يعمل لك communication عالي أو latency عن network أو congestion عن network، فبيبطئ لك التطبيق العملي بشكل عام، والمشكلة الثالثة الموجودة عنا إنه ممكن يكون لأنها هاي الـmicro services distributed على عدة-devices أو عدة-servers، فعملية نقل الـdata بين هاي الـservices، اللي هو المسج queuing، فيه خلل، بتضل preserve، يعني بتضل محفوظة عندك بالqueue، النظام scalable، على سيرفر واحد، كيف لي priority queue، وتوزعها عدد سيرفرات موجودة عندك، هلا، الاشي المهم كمان اللي موجود بهذا الasynchronous ،design، شو يعني asynchronous؟ يعني الclient ما بيهتم، بيرمد المسجات، المسجات كلها بتتخزن بالqueue، والسيرفر بيعملها عملية processing براحته، بين الclient والسيرفر، هذا بيسمح لك إنه الclient والسيرفر يشتغلوا، كل واحد بيشتغل بشكل منفصل، وهذا بيعمل لك كمان time

lakhasly.com © تم تلخيص النص بواسطة موقع لخصلي

decoupling، ببساطة، حتى لو صار فيه اختلاف في البودرات، ففيه كتير تطبيقات بتشتغل، بتنبنى على شكل message queuing، اللي هو الobserver pattern، ببساطة، لهذا الdesign. الآن شوف فكرته ببساطة، فكرته ببساطة إنه، لازم يكون عندي publisher، تمام، لازم يكون هدول الlisteners، تمام، أو extension له. بس هون بشكل عام بيحكي لك المسجات، بيسموها event، فهو نفس design حرفياً، message queuing in addition، إنه listener أو subscriber، أو بالنظام، اللي هو بيعمل receive للnotification، أو الobserver، لevents معين، لمجموعة من الevents، بيكونوا مسجلين already، عند مين؟ عند البublisher، فببساطة هو بيشبه نظام الobserver، أو لخلينا نحكي اللي بيعمل notification، بيكونوا عاملين registration عندهم، وأي event بيصير عند البublisher، مباشرة بنعمل notification للsubsystems اللي موجودين عندهم، طبعاً هدول الevents بتخزنوا داخل queue، عشان إذا صار فيه اختلاف البودرات بين السيستيم الأول، بتتخزن المعلومات كلها، فهو شبيه بالdesign الأول، أو بمعنى آخر extension إلو. هي اسمه pipe and filter architecture، الآن المطلوب منك عشان تكون عارف كيف طبيعة الماتريال هاي، على الأغلب بجيب لك نظام، طبعاً؟ ممكن أجي أحكي لك عندي airline system، وعندك sales system، sales system بدو يعمل notification، بكون حكيت لك description، برسم معين، على شكل على سبيل المثال publisher subscriber، أن الpublisher هو اللي بعمل publish لل event، والsubscriber هو اللي بستقبل notification، وترسم للdiagram الكورسبوند إلو، الabstract level تبعه كيف بيشتغل، ما أكيد تعرف كل واحد شو الفوائد، تمام؟ ونتيجة هذا الفلتر، بتعمل communication مع بعضها عن طريق الpipes، تمام؟ في كتير تطبيقات عملية بنيت على هذا الdesign، أشهر التطبيقات العملية اليونيكس كومند، احنا لما نكتب على اليونيكس كومند معين، عن طريق الكومند لاين، تمام؟ بتم ترجمته، وبعدين ترتبها، أو ترتب الcontents اللي داخلها، بتفلترها حسب. هذا فلوت، فتدخل على موضوع السيمانتيك أناليسيز، كل بلوك من هذه البلوكات أو كل فلتر عبارة عن برنامج كامل متكامل برمجياً، ستبرمج لغزيكال أناليسيز، برنامج يعمل التوكينيزيشن، برنامج يعمل السينتاكس أناليسيز، ستفهم برمجياً كيف فعلياً الابليكيشن اللي بتكتب وبنعمله كومبيليشن بشكل عام بأي لغة برمجي، بتطلع منها انستركشن جديد، اللي هي تطلع الانترمديت لانجويج، الانترمديت لانجويج اللي هي أسيمبلي لانجويج، قبل ما تحولها لماشين لانجويج اللي موجودة علينا، طبعاً فدول كل بلوك منهم عبارة عن أبليكيشن كامل متكامل، كتير كتير كبير بس أنا بحكي، ممكن تعمل كومبايلر بسيط مكون من كم جملة برمجية طبعاً، لكن بشكل عام الكومبايلرات بتكون فيها كومبليكسيتي عالية في بناء. يعني ما راح بكورس تبني كومبايلر لانسي ولا جافا، ما راح تبني لبرنامج بسيط دوميني سبيسيفيك لانجويج. اللي مطلوب منك انك تعرف انه هاي بعض التطبيقات العملية اللي بتستخدم مبدأ البيب اند فلتر. شو فيه أخرى تطبيقات ممكن تستخدم مبدأ البيب اند فلتر عندك الانتر برايس البزنس، ويركي فلو، ويركي فلو كيف بدك تدفع فاتورة معينة، بتأول إشي بتتم إصدار الفاتورة تقرأها، تحدد قديش مقدار البيمنت، يا إما بتشوف امتى آخر موعد لتدفع، وبعدين بتحط ريمايندر انك رح تدفع باليوم لفلاني، أو مباشرة بتدفع تاخد وصل نتيجة انك دفعت، يعني أمامك خيارين يا بتدفع بشكل مباشر يا بتأجل الدفع بمعنى أخر، كل بلوك فيهم فلتر بعملك أكشن معين، هلا بشكل عام، شو بتشوف فوائد ولميتيشن في هذا الديزاين؟ أول شغلة، أو في حدا عامل كومبايلر، وأكيد الليجزيكال أناليسيز ما رح يختلف، وين ما تحطه بأي لغة برمجينة، رح يكون البرنامج اللي بيعمل توكينيزيشن نفسه، يعني منفصل لهاي الفلترات عن بعضها، بس بتستخدم السيربيسز من بعضها، ففيري إيزي تعمل ريوز للابليكيشن. كتير مناسب بالويركي فلو، يعني أي اشي بتلاقي فيه ويركي فلو، هذا فيك ببساطة تروح تبنيه على شكل بايب أنت فلتر. لأي فلتر سهل جداً، انك تعمل وتضيف تعمله انتجريشن بالسيستم، ممكن يشتغل سيكوانشالي وممكن يشتغل كونكرانتلي، هذا سيكوانشالي وراء بعض بس هذا كونكرانتلي، لأنه انا ممكن اعمل فورك امشي بهذا الفلو، what the same moment امشي بهذا الفلو، فبشتغل على كل مبادئ الفوركينج ديزاين، ليه ببساطة سيكوانشالي او كونكرانتلي. هلا وين المشكلة في هذا الديزاين؟ المشكلة انه اذا زادت عدد البلوكات او الفلاتر، بزيد الoverhead عالسيستم، طبعاً يعني انت تخيل هذا النظام، ويبعطه للي بعده انت، هاي مشكلة من المشاكل الموجودة. المشكلة التاني في الديزاين انه، اذا كان الفلتر الاول مش على علم بال data format، رح يصير في عنا خلال في النظام ليه؟ لانه تخيل ال output اللي بيطلع من هذا البلوك عبارة عن json file، واللي بيستقبلوا البلوك التاني او الفلتر التاني عبارة عن xml file اذا اختلف الفورمات ال data طريقة وصف المعلومات بين الاطراف بصير عنا خلال بالنظام. فلازم يكون الطرفين agreed، اللي بتنتقل من طرف الى طرف اخر، لازم يكون متفق عليه مسبقاً، واقل بصير عنا خلال في عملية transformation، فمعناها انه انا اذا بدي اعمل reusing، لاي مقطع منهم لازم اكون على

lakhasly.com © تم تلخيص النص بواسطة موقع لخصلي

علم بنوع ال data structure و نوع ال data structure اللي طلع فورمات ال data، يعني تخيل معي هذا ال design، بتطلع لك data عشاكل tree، نوع ال data structure اللي طلع فورمات ال data فيه عشاكل tree، وهذا بيستقبل لكها عشاكل linked list، طبعاً هاي من اكتر ال limitation اللي ممكن الواحد يشوفها في ال pipe and filter architecture design. ليه لانه بتاخد ال data من excel sheet من csv file، تنظف ال data، مرات بيكون فيها corruption data، مرات بيكون فيها outliers، مثلاً تبها 40، فش درجة حرارة بتوصل 200 سلسيوس عصابين المدالة، فبدك تعمل cleaning لل data تشيل outliers، بعدين بتعمل data engineering، او بيسموها feature engineering، شو feature engineering معناها، في عندك عصابين للمثال، ال price لل بيوت، يعتمد على كم غرفة موجود فيه، او يعني قديش هو close من الـacilities، هلق فيه مجموعة من المعلومات موجودة، وانا اللي بدي اعمله prediction على سبيل المثال ال price based على input هذي الاربعة والخمسة، بدي اشوف سعر البيت هذي، بس فيه معلومة مش كتير مهمة او مش related لسعر البيت، فهذا الfeature engineering شو معناها، الفـeature engineering معناها انه بدي اشوف من هاي الفيتشارز اعمل reduction او اعمل remove للفيتشارز اللي مش مهمة، اعمل reduction او اعمل remove للفيتشارز اللي مش مهمة، لانه اذا بدي استخدم كل ال data ممكن كون حجمها ضخم جداً او الcolumns كلها، فبالتالي ممكن يتأثر على performance الموديل، فمنعمل اشي منسميه feature engineering، بعدين بتروح النتيجة بعد ما تعمل feature engineering لترينج للموديل، وبعدين بتروح لعملية validation الموديل، وبعدين بتروح للdeployment. فهدول كلهم style بنسميهم او اي machine learning approach بتشتغل على مبدأ اللي هو Shutterstock، pipe and filter style نيجي لاخر style عندنا اللي هو الbeer to beer architecture، اللي اخدوا نتويرك اكيد سمعوا بهذا اللي هو الbeer to beer او الpoint to point بسموه، بس انا بحكي هلأ عن الversion اللي بسموها mesh، المشة اللي معناها انت بتشبك الاجهزة بعض بعض بدون ما يكون عندك اي هيكلية داخلية infrastructure-less، ملهاش infrastructure معين، ملهاش structure معين. لكن ان وجد سيرفر في الdesign تخيل هذا السيرفر، for monitoring للapplication، اللي هو احنا كنا نسميه زمان semi-centralized architecture، لكن هذا distributed architecture بالعادة الclients بيحكوا بشكل مباشر مع بعض، يعني هذا كتير مستخدم في wireless networks فعليا، من الفوائد الكبيرة انه الاجهزة بتصير تستفيد من computational power والstorage الموجود عند الاجهزة والاخرى في الشبكات. هدول الwireless sensors مزروعين في بيئة معينة، تمام وبيستقبلوا هدول الsensors بيقرؤوا الenvironment، وبيدهم يعملوا computation based على الenvironment الموجود. يعني خلصت البطارية او صارت في low level، الفكرة من الdesign هذا الbeer to beer انه بيقدر يعمل عملية transfer للcomputation للاجهزة الاخرى، عشان يستفيد من الباور والstorage الموجود عند الاجهزة الاخرى فبتصير الاجهزة تتنقل عملية الcomputation بين بعضها. فهاي فائدة الbeer to beer بشكل عام، الاستفادة من الresources الموجود resources المقصود فيها الcomputation power زي البطاريات الstorage الhard disks and so on، عشان تعمل تكمل الcomputation كامل في الشبكة الموجودة. تمام كل node من هاي قد تكون server وقد تكون clients، وكل طبعا لاحظ الresources الموجودة في حدا بعمل consuming، في حدا بزودك بالresources، يعني ممكن هذا يكون جهاز ماعمل computation فممكن يعطي لهذا الجهاز power او يعني يطلب جهاز 13 من جهاز 12 يعمل الcomputation عند node رقم 12 على سبيل المثال ممكن هذا يخزن عند node رقم 12، فبصيروا يستفيدوا من الresources اللي موجودة عندنا وهذا بعمل collaboration في الapplication. زي هيك كتير بيستخدم الbit torrent اللي هو file sharing، عملية الdownload في الfile sharing ليه لانه ببساطة انت لما تيجي تسايب على جهاز ممكن هذا الجهاز تكون resources تبعته storage استهلكت، فممكن تستغل الstorage اللي عند 8 node تخزن عليها وتعمل sharing لهاي الملفات عليها، فهاي ببساطة مبدأ الpeer to peer architecture كتير مستخدم للnetworking، هذا من اكتر الarchitecture style اللي راح تشوفه في موضوع الnetwork بين الاجهزة، يعني اذا طبيعة التطبيق العامل يتبعك networking ببساطة راح تشتغل على هذا. هاي يا جماعة كل الاجهزة كل الأحوال مبلع يا علي مبلع، بس ما هو شو الفكرة كانوا عشان هيك كنت احكي لك عن الsemi-centralized السيرفر الرئيسي اللي بيكون كل المستخدمين هدول عاملين registration عندهم بهدف حفظ السيكوريتي اللي موجود، لكن فيه بروتوكولات بتضبط هذا الحكي. هاي جميع الarchitecture styles اللي ممكن انت تستغلها لبناء انواع مختلفة من التطبيقات العملية، starting من الlayered والmodel view controller المخصصين للweb، وانطلاقا باي نوع

lakhasly.com © تم تلخيص النص بواسطة موقع لخصلي

اخر من التطبيقات العملية زي compiler زي networking system، يعني فيه كتير يعني هون الlayered والmodel حكينا web microservice اذا بدك تبني التطبيق وتستغل فكرة scalability، الpeer to peer متخصص اكتر في التطبيقات الnetworking اللي فيها networking. فهاي معظم الarchitecture styles very common المستخدمة جدا في بناء التطبيقات العملية up to date يعني لحد ايامنا هاي كلها بتستخدم في بناء التطبيقات العملية. بيركز على الdesign principles الsolid principles، الcoupling الdecoupling الproperties الخاصة في الdesign. يعني we assume اننا جمعنا requirements، we assume عملنا الdesign عملنا الimplementation وهلأ بنروح لمرحلة التستين. طبعا في كتير شغلات انا رح اعمل skip عليها ان شاء الله لحق اخلصها خلال الفصل بس رح اروح هيلا جنب على الشغلات الاساسية عشان نضبط الماتريال تكون اخدت كل الافكار الاساسية بالمدى طبعا. فهلأ رح نروح لموضوع اللي بيركز بشكل عام اللي بيركز على the quality للكود اللي انت كتبه، احنا بلشنا وشرحنا شق معين يعني انا هلأ اللي بهمني how to improve the quality للكود اللي عنده، Shutterstock اليوم بدي احكي عن موضوع اخر بالتستنج اسمه test-driven development هذا موضوع جدا مهم وموضوع الstatic analysis رح يكون بفيز رقم 2 بالمشروع، خلينا نروح لموضوع test-driven development تقريبا سهل جدا هذا الموضوع قريبا انت عامله بس بطريقة عكسية، مفهوم test-driven development ببساطة كيف تكتب التست كيسز قبل ما تبني اي سطر كود قبل ما تعمل الproduction code طبعا، احنا رح نرجع نستغل الجاي unit نفسها لتطبيق هذا المبدأ اللي هو اسمه test-driven development.

lakhasly.com © تم تلخيص النص بواسطة موقع لخصلي